MASTER EN SCIENCES INFORMATIQUES À FINALITÉ SPÉCIALISÉE EN SOFTWARE ENGINEERING

METAssistant

Un outil d'évaluation de programmes informatiques destiné aux étudiants

Colart, Thomas; Vaneck, Quentin

Award date:
2021

Awarding institution:
Universite de Namur

METAssistant

Un outil d'évaluation de programmes informatiques destiné aux étudiants

COLART Thomas
VANECK Quentin
UNIVERSITÉ DE NAMUR
Faculté d'informatique

Année académique 2020-2021

Maître de stage : Benoît VANDEROSE
Promoteur : _____ Benoît VANDEROSE
(Signature pour approbation du dépôt - REE art. 40)
Co-promoteur : Benoît FRÉNAY

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques

Remerciements

Nous tenons à remercier sincèrement nos deux promoteurs, les professeurs Benoît Frénay et Benoît Vanderose pour leur suivi, leurs retours et leurs précieux conseils pour le mémoire de fin d'études comme pour l'article soumis.

Notre reconnaissance se tourne aussi vers le corps enseignant de l'UNamur dans son ensemble ainsi que dans celui de l'IESN, lieu de notre bachelier en informatique de gestion.

Merci également à nos familles et amis qui nous ont soutenus durant ces années d'études. Et un merci particulier à Céline, Déborah, Quentin et Lloyd pour leur aide et leurs conseils.

Résumé

Les études dans le domaine de l'informatique sont de plus en plus populaires, et les enseignants doivent faire face et s'adapter au nombre croissant d'étudiants. Alors que les petits groupes permettaient plus d'interactions entre les enseignants et les étudiants, le surnombre qui en résulte supprime la proximité et oblige les enseignants à passer moins de temps avec chaque étudiant. Par conséquent, l'étudiant peut rapidement se sentir submergé et désemparé face à la difficulté du cours. Cet article propose une solution qui vise à réduire les abandons dans les cours de programmation. Elle offre un retour précis sur la qualité du code Python des étudiants afin d'approfondir leur compréhension, ainsi qu'une interface ludique pour stimuler leur intérêt pour la programmation. Cette solution développée sous le nom de "METAssistant" a deux objectifs. Elle permet aux étudiants de l'utiliser pour évaluer leurs programmes et obtenir un retour précis, et aux enseignants d'avoir un aperçu de la compréhension de la matière par leurs étudiants.

Mots clés : Cotation automatique, Auto-évaluation, Qualité logicielle, Analyse statique, Gamification.

Abstract

Computer science studies are more and more popular, and teachers must face and adapt to the increasing number of students. Whereas small groups allowed more interactions between teachers and students, the resulting overcrowding takes away closeness and forces teachers to spend less time with each student. Therefore, the student can quickly feel submerged and helpless against the difficulty of the course. This paper proposes a solution that aims to reduce drop-out in programming courses. It offers an accurate feedback on the quality of students' Python code to deepen their understanding, together with a playful interface to boost their interest in programming. This solution developed under the name of "METAssistant" has two objectives. It allows students to use it to evaluate their programs and to get an accurate feedback, and teachers to have an overview of the understanding of the matter by their students.

Keywords: Automated grading, Self-assessment, Software quality, Static analysis, Gamification.

Table des matières

Table des figures

1 Introduction

L'intérêt de ce point est de fournir toutes les clés de compréhension nécessaires afin de comprendre l'objectif du travail ainsi que le but de la solution qui sera mise en place. Pour cela, vont être décrits le contexte de l'étude, les objectifs du projet, la question de recherche sur laquelle va se baser ce mémoire ainsi que le plan global de ce dernier.

Ce mémoire a fait l'objet d'une soumission au troisième workshop international EASEAI à la conférence ESEC/FSE 2021. L'Annexe 3 contient l'article soumis.

1.1 Contexte

À l'université de Namur, il y a de plus en plus d'étudiants en informatique. La tâche des professeurs et assistants en est donc alourdie, que ce soit pour la correction d'exercices ou pour aider les étudiants à comprendre au mieux la matière. De plus, l'informatique est un domaine compliqué et en constante évolution, et la programmation peut vite devenir une tâche laborieuse pour les nouveaux arrivants.

Dans ce cadre, un mémoire à précédemment été réalisé afin de trouver une réponse à ces problèmes. La solution proposée (CHATssistant[1]) - l'existant du point de vue de ce mémoire - est un chatbot permettant à un étudiant utilisant Visual Studio Code[2] de l'aider dans la rédaction de son code et de l'exécuter afin de vérifier la pertinence de celui-ci, c'est-à-dire qu'il n'y ait pas d'erreurs de compilation ou de syntaxe. D'un autre côté, un professeur ou assistant peut accéder, via une plateforme dédiée, aux exercices des étudiants et à certaines statistiques afin d'assurer un suivi complet au cours de son apprentissage.

1.2 Objectifs

L'objectif est d'apporter des améliorations au système de vérification du code. De part ces améliorations, la tâche est double. Tout d'abord il convient de permettre aux étudiants une meilleure compréhension des mécanismes de la programmation en leur donnant un feedback clair et précis des erreurs commises. Le but est d'offrir aux étudiants un retour expressif et ludique sur les exercices qu'ils ont réalisés. C'est à dire noter de la façon la plus formelle possible la qualité d'un programme pour que l'étudiant puisse se situer dans son apprentissage, le tout dans un environnement attrayant et propice à booster l'intérêt des élèves pour la programmation. Ensuite, il est nécessaire de réduire le plus possible la charge de travail des professeurs et des assistants. Le but étant d'alléger leur travail de correction en déléguant un maximum les tâches automatisables pour permettre à l'étudiant d'être plus autonome, sans pour autant le laisser, livré à lui-même.

Aussi, il existe déjà des outils d'analyse permettant d'évaluer un code de diverses manières. Mais ceux-ci manquent souvent de réussite lorsqu'il s'agit d'analyser des codes non-compilables et syntaxiquement incorrects d'étudiants de 1ère année. Ce travail tente d'apporter une solution à ce problème en associant un outil d'analyse statique avec un algorithme de correction pour les lignes erronées afin de permettre l'exécution de l'analyse sans problèmes.

[1] Solution du mémoire CHATssistant https://github.com/jacquant/botlearn.
[2] Visual Studio Code : éditeur de code développé par Microsoft.

1.3 Question de recherche

Comment fournir une cotation et un retour compréhensible sur la qualité d'un code remis par un étudiant dans le cadre du cours de « Principes de programmation » ?

Sur base de cette question de départ et afin d'y répondre au mieux, ont été établies plusieurs sous-questions auxquelles notre travail devra répondre :

- Quels facteurs sont à prendre en compte pour l'évaluation d'un code et comment mettre en place cette évaluation ?
- Comment utiliser des outils d'analyse statique pour automatiser la cotation et le feedback des programmes, même lorsque le code peut ne pas être exécutable et ne pas respecter la syntaxe Python ?
- Comment inciter les étudiants à améliorer leurs compétences en programmation par le biais de la formation continue ?

1.4 Plan du mémoire

Le but de ce mémoire est d'améliorer le confort d'apprentissage des étudiants en leur proposant un outil adapté à leurs besoins. À travers ce travail, sera décrite la solution ainsi que les différents choix qui l'auront structurée et leurs influences.

Le projet de l'an passé (l'existant) offre à l'élève une aide syntaxique pour son apprentissage. L'objectif ici, est de fournir un outil en plus afin de faciliter encore plus l'apprentissage d'une discipline aussi pointue que la programmation. Pour ce faire, un nouvel outil est mis en place et celui-ci se veut complémentaire aux éléments déjà présents dans cet écosystème (le chatbot et Webcampus notamment).

La section 2 concerne l'état de l'art et permet d'approfondir certaines connaissances qui s'avèrent nécessaires au développement de la solution, au travers d'une revue de littérature. La section 3 décrit toute la partie conception du projet, avec la description des différents choix faits et les raisons de ceux-ci. C'est également dans cette section que sera abordée la contribution principale de ce projet. Ensuite, la section 4 développe les tests effectués afin de juger de la pertinence de la solution et s'assurer qu'elle remplit bien son rôle.

Pour finir, la section 5 vient clore ce travail avec diverses pistes d'améliorations et travaux futurs envisageables. En effet, le stage durant approximativement 15 semaines, il n'est pas possible de combler tous les aspects souhaités. C'est pourquoi des priorités ont été définies, le plus de thématiques possibles sont traitées, mais il n'était pas possible de tout implémenter.

2 État de l'art

Ce chapitre consiste en une revue de littérature permettant de cerner correctement les différents enjeux du mémoire. Au travers d'analyses et de liens avec différents concepts vus pendant le cursus, l'objectif est de voir différentes méthodes d'évaluation et comprendre les facteurs qui influencent la cotation d'un code. Pour ce faire, les étudiants de Bloc 1 et plus spécifiquement ceux suivant le cours « *INFOB131 Introduction à la programmation* » servent de référence au niveau des besoins générés et de la pertinence de la solution. Ce chapitre est divisé en 3 points, importants à développer afin de bien comprendre les enjeux et se renseigner sur les méthodes et outils qui permettent de définir une solution adéquate et optimale.

Dans un premier temps, la section 2.1 décrit les facteurs qui permettent d'évaluer la pertinence d'un code en tenant compte de tous les éléments impliqués ainsi que leurs impacts sur l'étudiant. Différents types d'évaluation notamment sont décrits tel que l'évaluation sommative et formative, le travail en groupe et ses bienfaits, ainsi que l'évaluation systémique, son rôle croissant et son implication dans la société d'aujourd'hui que ça soit dans l'entreprise ou dans l'éducation en général. Des méthodes d'évaluation de programmes informatiques sont également développés à la fin de cette section. Ensuite, la section 2.2 démontre les avantages pédagogiques de la gamification et ses applications spécifiques à la programmation dans le but de fournir une correction ludique et motivante pour l'étudiant. Pour finir, la section 2.3 présente différentes solutions existantes proches du but recherché et qui peuvent éventuellement servir de base d'appui pour la solution.

2.1 Pertinence de l'évaluation

Dans le milieu de la pédagogie, il existe plusieurs méthodes d'évaluation adaptées en fonction du contexte et de l'objectif poursuivi. Les évaluations formative et sommative notamment sont développées dans ce point ainsi que le modèle systémique qui est une autre approche intéressante à explorer. La question de l'évaluation des membres d'un groupe est également abordée dans cette section.

2.1.1 Évaluation formative

L'évaluation formative a pour but d'informer à la fois l'étudiant et le professeur. Elle consiste en une évaluation continue de l'élève permettant de situer son niveau de compréhension d'une matière spécifique. Cette façon d'évaluer ne se limite pas forcément à un échange entre un professeur et un étudiant, elle peut également être faite sous la forme d'autoévaluation par exemple. L'autoévaluation peut se définir comme un ensemble de pratiques d'autorégulation en trois parties à savoir : le choix des objectifs et des niveaux à atteindre, la recherche et l'utilisation du feedback et le choix des moyens d'actions. La personne s'autoévaluant fixe donc ses propres objectifs ainsi que la façon d'y parvenir [**Laveault, 2009**].

La principale caractéristique de cette méthode est donc de permettre à l'étudiant de tester sa connaissance de la matière au cours de l'année, afin que le professeur puisse adapter son cours en ajoutant des activités qui vont venir renforcer la maitrise de la matière. Pour ce faire, l'évaluation n'est pas associée à une cotation, un pourcentage ou tout autre forme de quantification. C'est essentiellement une étape précédant l'évaluation sommative et qui va préparer l'étudiant à cette dernière. La Figure 1 exprime ce processus à tous les étages.

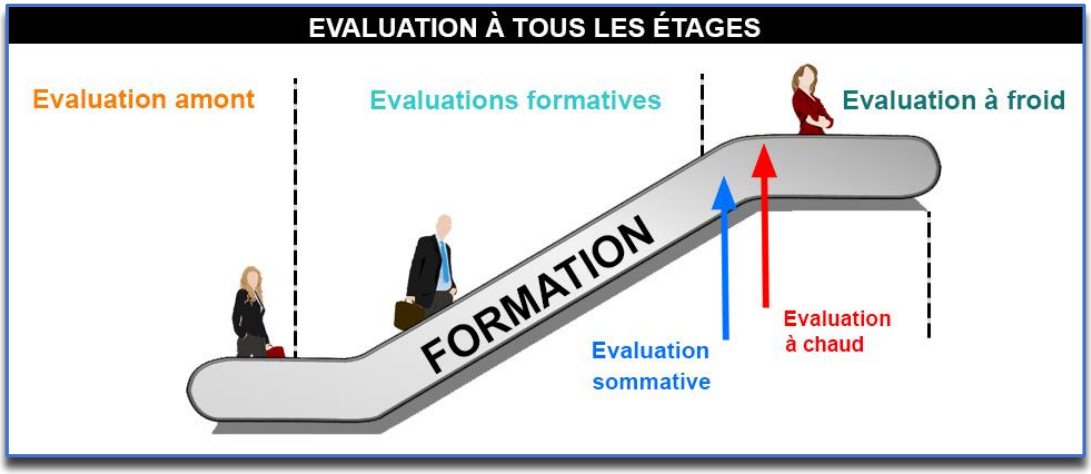


Figure 1 : L'évaluation au cours de la formation, par E. Gabas[3]

2.1.2 Évaluation sommative

L'évaluation sommative consiste en une attestation qui certifie que les compétences sont bien maitrisées. Celle-ci provient du professeur à destination de l'élève et doit être juste et équitable car elle vise à quantifier la connaissance de l'étudiant par rapport à un sujet [**Morissette, 2009**].

Pour garantir l'équitabilité du test, celui-ci peut prendre différentes formes telles l'examen écrit, le questionnaire à choix multiples, la dissertation orale... Certaines modalités d'évaluations se prêtent mieux en fonction du contexte, la matière propre à un cours peut parfois être restituée de façon plus claire si le type d'examen choisi est le bon.

Le retour fait par le professeur à un étudiant dépendra également de cette modalité d'évaluation. Si l'examen est oral, les commentaires du professeur le seront certainement aussi, ce qui permet d'avoir un retour direct, mais parfois moins objectif si les critères de cotation ne sont pas bien définis. Un questionnaire à choix multiple, du fait de ses questions plus fermées, sera probablement moins fourni en commentaires du professeur. Un examen écrit est souvent plus objectif et semble être l'approche la plus pédagogique pour que l'étudiant comprenne bien la notation qui lui a été attribuée. Le retour fournit par le professeur est ainsi capital car l'étudiant doit être capable de mettre des mots sur ses faiblesses afin d'y remédier. De cette façon, il sera capable de corriger d'éventuelles lacunes et de progresser dans son apprentissage [**Scallon, 1988**].

Un bon apprentissage passe donc par une combinaison de différents types d'évaluations de façon que l'étudiant puisse constamment évaluer ses capacités avant de procéder à un test concret. La Figure 2 montre un exemple de processus pouvant être utilisé dans le cadre de l'enseignement, qui alterne entre des évaluations formatives et sommatives de manière à garantir une meilleure évaluation des connaissances de l'étudiant.

[3] Evaluation à tous les étages, repris depuis https://1day1learn.com/levaluation-tous-les-etages-avec-le-digital-learning/.

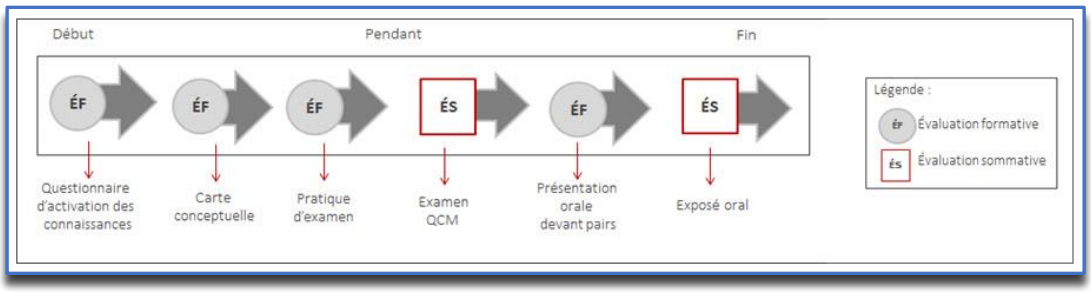


Figure 2 : Cycle d'évaluation formative et sommative, par Lussier et Bélanger (2009)[4]

L'évaluation aujourd'hui consiste à donner une note finale qui va attester du degré de réussite d'un travail, cependant cette note ne semble pas toujours pertinente. En effet, il est très souvent possible de décomposer un travail en plusieurs critères qui vont pouvoir être évalués individuellement. Cette façon de faire permet de déterminer des forces et des faiblesses propres au travail, mais qui seront peut-être masquées par la généralisation au moyen d'une note globale.

2.1.3 Évaluation de groupes

L'apprentissage en milieu scolaire repose énormément sur l'interaction sociale et parfois spécifiquement sur la notion de groupe. Il y a souvent des problèmes au niveau de l'évaluation dans un groupe, notamment car tous les étudiants n'auront pas forcément la même implication. Ainsi, donner une note semblable à tous les membres serait perçu comme injuste car la motivation et le travail fourni serait différent pour chacun d'entre eux.

Deux questions entrent alors en collision :

(1) « Faut-il évaluer la contribution de l'étudiant au travail ? »
(2) « Faut-il évaluer le groupe dans son intégralité sans dissocier les membres ? »

Bien qu'il soit possible de faire un mix, en prenant en compte à la fois le groupe et l'individu, l'approche souvent préférée est la première car l'étudiant doit-être jugé pour sa contribution personnelle avant tout. Un bon moyen d'évaluer un travail de groupe est de procéder à une évaluation entre les pairs du groupe, chacun y trouvant ainsi un sentiment d'équité et un accroissement de son investissement dans le groupe [**Moccozet, Tardy, Opprecht et Michel, 2013**].

2.1.4 Modèle systémique

Le modèle systémique trouve une de ces définitions grâce à l'AFSCET[5] qui décrit cela comme étant « une nouvelle discipline qui regroupe les démarches théoriques, pratiques et méthodologiques, relatives à l'étude de ce qui est reconnu comme trop complexe pour pouvoir être abordé de façon réductionniste, et qui pose des problèmes de frontières, de relations internes et externes, de structure, de lois ou de propriétés émergentes caractérisant le système comme tel… ».

L'approche systémique est un modèle plus tourné vers l'extérieur et non centré sur l'individu. L'accent est mis ici sur la communication et les interactions entre les concernés afin

[4] Cycle d'évaluation formative et sommative, repris depuis https://www.enseigner.ulaval.ca/ressources-pedagogiques/l-evaluation-formative-et-sommative.
[5] AFSCET : Association Française des Sciences des Systèmes Cybernétiques, Cognitifs et Techniques.

de déterminer comment une personne est influencée par tous les éléments d'un milieu et comment elle influence ce milieu. Les problèmes affectant un individu peuvent ainsi se répercuter sur un système entier par exemple [**Karsenti, 2018**]. Ce modèle peut s'appliquer à différents milieux (industrie, familial...). Son application dans l'enseignement est expliquée ici afin de comprendre au mieux son fonctionnement.

2.1.4.1 Le modèle systémique dans l'enseignement

Au niveau scolaire, ce modèle peut prendre plusieurs approches. L'une d'elle est par exemple de se focaliser sur les éléments qui entourent l'étudiant c'est-à-dire sa situation sociale et familiale notamment. C'est donc de se concentrer sur tous les éléments qui pourraient avoir une influence sur sa réussite scolaire [**Berlioz, 2007**].

La Figure 3 montre un ensemble de relations prise en compte dans la mise en place d'un modèle systémique. Ce modèle est celui d'une évaluation, il concerne à la fois des éléments personnels en lien avec l'étudiant (relations, famille...), mais aussi des éléments davantages liés à l'éducation (politique d'établissement, plan d'études, objectifs...)

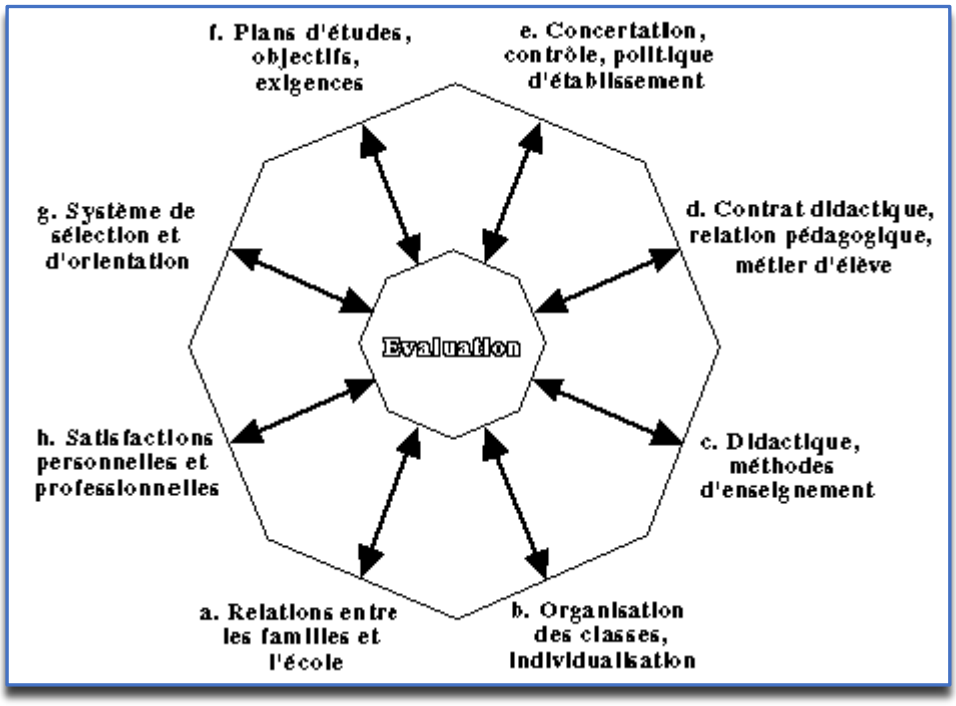


Figure 3 : Ensemble de relations constituant un modèle systémique d'évaluation[6]

Il est néanmoins difficile d'effectuer une analyse aussi précise pour tous les étudiants car les situations personnelles varient pour tous. Il est donc préconisé de mettre en place un canevas permettant de traiter au mieux les différentes problématiques auxquelles l'étudiant fait face. Une notion importante dans ce canevas est celle de l'encadrement. L'objectif de ce point est de mettre en place un système hiérarchique à 3 niveaux avec l'étudiant, l'enseignant et un intervenant qui serait chargé de surveiller le processus d'interaction. Ce système d'encadrement démontre qu'un professeur doit trouver un équilibre entre l'adaptation à ses élèves et la rigidité par rapport à la matière qu'il aborde ainsi que les objectifs à atteindre. Cet équilibre va favoriser

[6] L'évaluation au centre d'un octogone de forces, repris depuis https://www.unige.ch/fapse/SSE/teachers/perrenoud/php_main/php_1993/1993_19.html.

l'autonomie chez les étudiants qui vont assimiler les buts à atteindre et vont adapter leur comportement pour s'y tenir [**Curonici et McCulloch, 2004**].

Cette notion de canevas peut s'apparenter à un système mis en place dans le but de favoriser l'apprentissage ou plus généralement l'enseignement. L'ensemble de ressources et d'outils est orientés vers les acteurs de ce système, les apprenants c'est-à-dire dans ce cas-ci, les étudiants. L'approche menée par un professeur dans le cadre d'un cours doit donc être équilibrée entre les objectifs à atteindre et une adaptation aux élèves. Un système ne doit donc pas être ni trop rigide ni trop peu, mais doit plutôt trouver un compromis de façon que tous les étudiants puissent apprendre de manière équitable. De cette façon, une propriété implicite du système doit être sa flexibilité. En effet, en fonction des résultats acquis par les étudiants et de l'évolution de ceux-ci, le système doit tendre vers des ajustements afin de garantir un apprentissage amélioré au fil du temps. Pour ce faire, analyser les résultats est une bonne façon de voir si le système mis en place fonctionne, tout comme prendre en compte le feedback des élèves est également conseillé [**Elkamoun, 2014**].

2.1.5 Évaluation basée sur les tests

De nombreux outils automatisés d'auto-évaluation ou de notation sont actuellement basés sur l'utilisation de cas de test. Cette méthode se base sur la définition de tests pour différents exercices, qui vont pouvoir être exécutés en temps voulu afin d'évaluer la fonctionnalité d'un code. Le Tableau 1 décrit divers avantages et inconvénients de cette méthodologie.

Tableau 1 : Avantages et inconvénients de l'évaluation de codes basée sur les tests

Avantages	Inconvénients
Il peut vérifier l'aspect dynamique d'un programme, c'est-à-dire son comportement et son bon fonctionnement	Il ne tient pas compte d'autant d'éléments que le ferait un enseignant, comme les bonnes pratiques (conventions, lisibilité, etc.)
Il fournit toujours un feedback pertinent dans le cadre de l'exercice	Les tests doivent être planifiés à l'avance pour chaque exercice

2.1.6 Évaluation orientée qualité

L'analyse d'un programme mis en comparaison avec d'autres candidats se base sur diverses caractéristiques telles que la fonctionnalité, le coût, la part de marché, le support, la maintenabilité, la fiabilité, les performances, l'évolutivité, l'utilisabilité, la sécurité, la flexibilité, et les questions juridiques/de licence [**Wajeeh, 2006**].

L'évaluation orientée qualité consiste à utiliser la mesure du logiciel pour évaluer un ensemble de critères de qualité. La pertinence de ceux-ci varie en fonction du contexte et de l'objectif du code évalué. Ces critères de qualité se basent en partie sur la norme ISO 9126, qui définit un modèle de qualité général applicable pour tout programme informatique. La norme ISO/IEC 25000 vise à fournir un framework[7] pour l'évaluation de la qualité des produits logiciels en regroupant et mettant à jour les normes existantes relatives à la qualité logicielle

[7] Framework : infrastructure conçue pour aider les développeurs en fournissant un ensemble d'outils et un cadre de travail, un canevas à suivre qui va augmenter la productivité et ainsi baisser les coûts de construction d'un projet.

[**ISO, 2014**]. Elle redéfinit l'ISO 9126 avec notamment l'ISO 25010 qui redéfinit le modèle de qualité standard.

Figure 4 : Critères de qualité définis par la norme ISO/IEC 25010[8]

La Figure 4 décrit chaque critère de ce modèle, à savoir la fonctionnalité, la performance, la compatibilité, l'utilisabilité, la fiabilité, la sécurité, la maintenabilité et la portabilité. Chacune possède des sous-caractéristiques telles que l'interopérabilité pour la complexité ou encore l'adaptabilité pour la portabilité.

Ces attributs peuvent eux même être définis par des facteurs de qualité [**Martin et Shafer, 1996**]. La Figure 5 schématise la division de quatre critères, maintenabilité, évolutivité, portabilité et descriptivité, par rapport à leurs facteurs de qualité. S'y retrouvent la cohérence, l'indépendance, la modularité, la documentation, l'auto-descriptivité, le contrôle d'anomalies et la simplicité de conception.

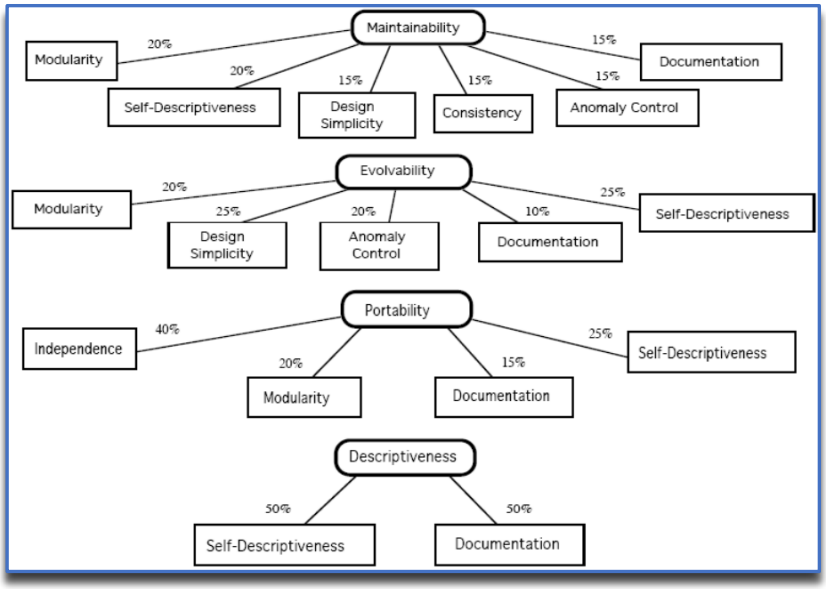


Figure 5 : Exemple de division de critères de qualité en facteurs de qualité, reproduit depuis Martin et Shafer, 1996

Les facteurs de qualité peuvent varier en fonction des besoins. Ils sont définis par des mesures et différents attributs estimés pertinents pour évaluer la qualité d'un code et chaque facteur peut être jugé plus ou moins important.

Comme l'ISO 9126, en plus du modèle de qualité, la norme ISO/IEC 25000 apporte également des métriques de base permettant d'évaluer la qualité d'un programme. Il amène aussi un processus pour la construction d'une mesure décrit à la Figure 6.

[8] Description de la norme ISO/IEC 25010, reproduit depuis https://iso25000.com/index.php/en/iso-25000-standards/iso-25010.

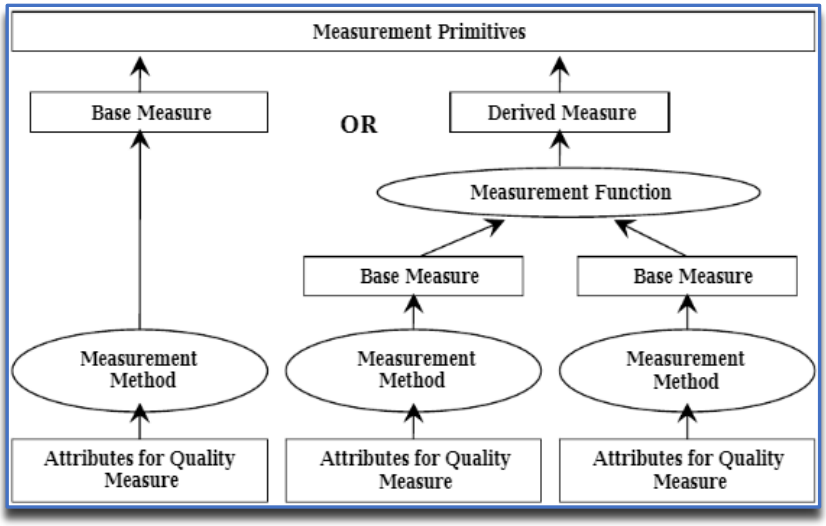


Figure 6 : Processus de construction d'une mesure primitive proposé par l'ISO 25000, reproduit depuis Abran, Al-Qutaish, Desharnais et Habra, 2005

Cette figure montre comment définir une mesure primitive, qui se rapporte aux facteurs de qualité cités précédemment. Cette mesure est d'abord constituée des attributs de base du programme source desquels vont découler des métriques à la suite de l'application de méthodes de mesure. Ces mesures ou les mesures dérivées, obtenues par la combinaison de mesures via divers calculs, forment les métriques qui vont être utilisées pour l'évaluation des critères.

Les mesures dérivées sont souvent plus pertinentes que des simples mesures, car les simples sont souvent basées sur des échelles absolues, moins pertinentes que des échelles d'intervalles ou de ratio obtenues en les dérivant. En effet, l'échelle absolue comprends des éléments uniques, comparables uniquement à eux même. Elle n'est pas transformable vers une autre échelle et permet de compter sans plus. Il y a quatre différentes échelles de mesure reconnues et utilisées dans la littérature, c.-à-d. nominale, ordinale, intervalle et ratio, allant de la plus transposable à la plus adaptée pour les calculs [**Stevens, 1946 ; Fenton et Bieman, 2014**].

La Figure 7 montre un exemple de mesures primitives proposées par l'ISO 25021 utilisables afin de dériver des mesures.

MP Class Name	MP Name
External Metrics	Time
	Number of Functions
	Number of Faults
	Number of Data
	Number of Operations
	Number of Test Cases

Figure 7 : Exemples de métriques de l'ISO 25021, reproduit depuis Abran, Al-Qutaish, Desharnais et Habra, 2005

À noter que la plupart des métriques proposées par l'ISO sont des simples mesures d'échelle absolue et ne sont donc pas les plus pertinentes pour évaluer la qualité d'un code. Néanmoins, elles restent intéressantes et donnent un aperçu et une base pour les personnes débutant la qualité logicielle.

Quality Group Name	Quality Measure Name
Internal Quality Measures	Functional Adequacy
	Precision
	Restartability
	Physical Accessibility
External Quality Measures	Computational Accuracy
	Access Controllability
	Operational Consistency
	Installation Flexibility
Quality in Use Measures	Task Completion
	Productive Proportion
	Discretionary Usage

Figure 8 : Exemples de mesures de qualité pour des évaluations interne, externe et de qualité à l'usage, reproduit depuis Abran, Al-Qutaish, Desharnais et Habra, 2005

La Figure 8 montre des mesures de qualité proposées par la norme ISO 25000. Elles sont divisées en trois catégories, les mesures internes, externes et de qualité à l'usage. Les premières concernent surtout les spécifications et le code tandis que les secondes concernent plus les fonctionnalités en elles-mêmes. Les qualités à l'usage reprennent majoritairement ce qui touche à l'utilisabilité et la sécurité.

Les concepts de mesures primitives et mesures de qualité sont propres à l'ISO 25000, mais ils ne respectent pas la terminologie classique et amènent de l'ambiguïté quant au mapping avec l'ISO 9126.

Un modèle de mesure de qualité a donc été fait avec l'aide de l'ISO 15939 et est représenté en Figure 9 pour clarifier ces concepts en utilisant les termes de métrologie basiques ainsi que fournir une description plus propre du modèle de mesure [**Abran, Al-Qutaish, Desharnais et Habra, 2005**].

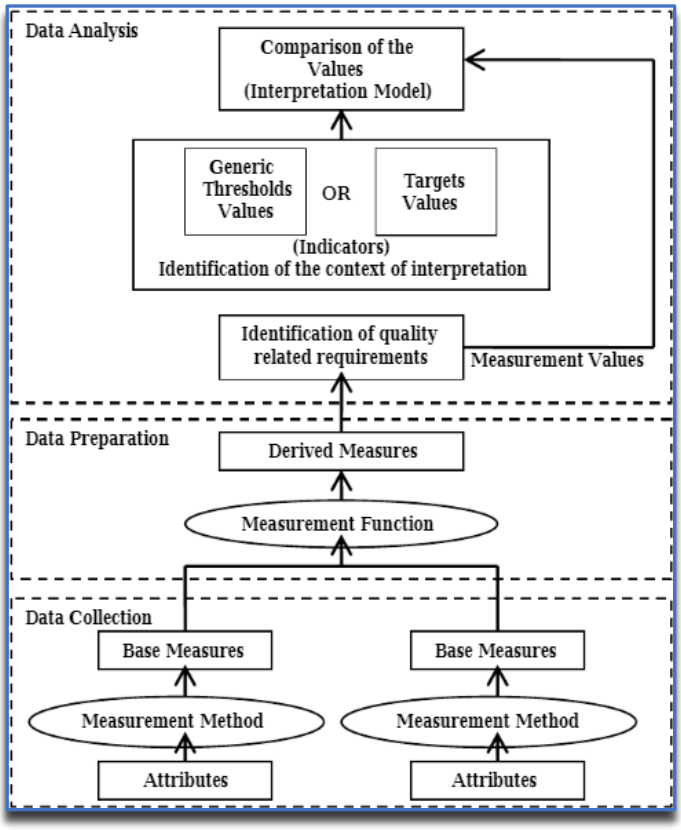


Figure 9 : Modèle proposé de mesure d'informations pour l'évaluation d'attributs de qualité, reproduit depuis Abran, Al-Qutaish, Desharnais et Habra, 2005

La Figure 9 représente ce modèle de mesure où la notion de mesure primitive a été enlevée pour ne parler que de mesures dérivées/de base. L'évaluation proposée est divisée en 3 étapes. Premièrement, la collection de données d'où vont être récupérées les mesures de base calculées depuis des attributs. Deuxièmement, la préparation des données, où les mesures de base sont dérivées. Et finalement, l'analyse de données dans laquelle est identifiée la qualité relative à la mesure, et où les valeurs obtenues sont comparées et interprétées à l'aide d'indicateurs. Cette méthodologie comporte également divers avantages et inconvénients cités dans le Tableau 2.

Tableau 2 : Avantages et inconvénients de l'évaluation de codes orientée qualité

Avantages	Inconvénients
Processus automatisé, pas besoin de prédéfinir explicitement les tests pour chaque exercice	La définition de toutes les mesures peut être une tâche longue et fastidieuse
Prend en compte de nombreux critères différents pour une évaluation plus précise	
Basé sur une norme réputée élaborée par des experts	

2.2 Gamification

La gamification est le terme anglais pour « Ludification » qui est l'application de mécanismes ludiques à ce qui ne relève pas du jeu [**Robert, 2020**]. C'est un principe qui consiste à appliquer des mécaniques propres au jeu et au divertissement en général afin de rendre une tâche plus attrayante [**Fels et Seaborn, 2015**]. Le fait de rendre une situation plus ludique pour un utilisateur a plusieurs buts.

Tout d'abord, au niveau de l'engagement de l'utilisateur, celui-ci peut se retrouver plus investi dans une tâche s'il a un but, un objectif à atteindre, un classement... Pour motiver l'utilisateur à s'investir, il est intéressant de définir des récompenses en tout genre pour des objectifs atteints. Cela permet de motiver l'utilisateur pour s'assurer qu'il se donne à fond dans la tâche en question.

L'aspect social est également important. Le contact avec d'autres utilisateurs permet la coopération et l'entraide ce qui offre la possibilité à tout un chacun de s'investir et de parfois trouver des solutions innovantes à une problématique. Ces effets bénéfiques peuvent aussi s'appliquer à la compétition. Par exemple, un classement, des récompenses... peuvent motiver un utilisateur à progresser dans le but d'être le meilleur [**Deterding, Dixon, Khaled et Nacke, 2014**].

La gamification est aujourd'hui un moyen utilisé dans beaucoup de systèmes connus. « LinkedIn » par exemple, possède un léger aspect de gamification encourageant à compléter son profil à l'aide d'une barre de chargement qui se remplit au fur et à mesure de la progression comme le voit sur la Figure 10. C'est ici de la gamification assez soft, mais pertinente, qui éveille chez l'utilisateur le besoin de remplir cette jauge pour en arriver au terme.

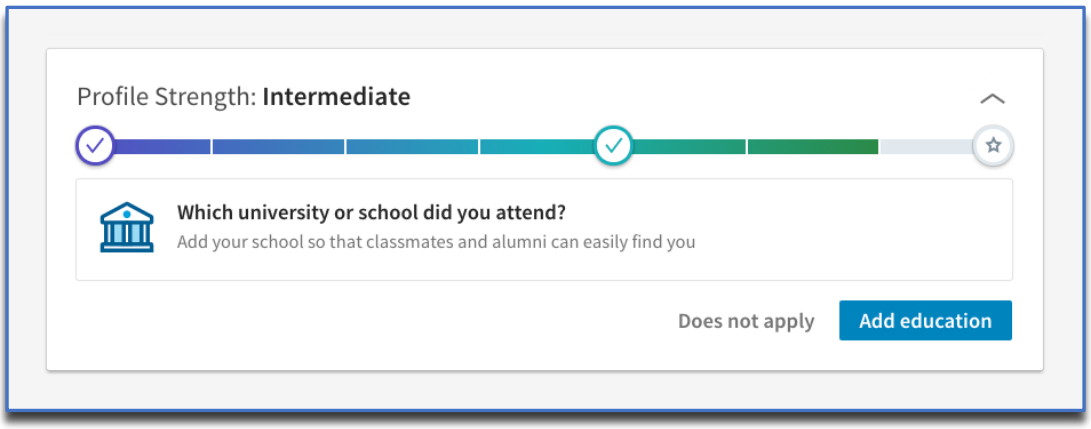


Figure 10 : Aspect de gamification de l'édition du profil LinkedIn, reproduit depuis LinkedIn

« Duolingo » va encore plus loin, basant tout son système sur la gamification. Cette application permet d'apprendre des langues au travers de divers exercices qui se débloquent au fur et à mesure de l'avancée de l'utilisateur. Elle lui permet également de se confronter avec d'autres personnes dans un classement hebdomadaire et de comparer les profils afin de situer son niveau d'apprentissage par rapport à d'autres utilisateurs. La Figure 11 illustre les différents aspects de gamification rendu possible par l'application tant au niveau du style de l'application, que dans la notion de classement et d'objectif qu'elle propose.

Aussi, il est possible de personnaliser son expérience via un petit magasin et une monnaie interne débloquée au fil de l'expérience acquise. Finalement, les différents éléments qui composent le jeu font l'objet de défis menant l'utilisateur à s'impliquer davantage afin de les réaliser.

Figure 11 : Aspects de gamification au sein de l'application Duolingo[9]

2.2.1 Évaluation des groupes via la gamification

La gamification peut également être utilisée pour dynamiser l'évaluation des étudiants dans le cadre d'un travail de groupe. La récompense reste un élément central, sur base de bons résultats, l'élève se voit attribué divers éléments qui vont attester de sa bonne implication dans

[9] Reproduit depuis l'application Duolingo

le travail (badges, trophées, collectables en tout genre...). Les membres du groupe peuvent être récompensés pour plusieurs raisons comme la qualité du travail rendu, leur implication, l'originalité... Ces différents éléments peuvent être adoptés en fonction de la situation d'évaluation, mais ils partagent tous un but commun : augmenter la motivation et l'implication de l'élève dans le travail. Cette motivation est intrinsèque à la notion de groupe, l'étudiant par la motivation personnelle que lui inspire le travail, sera plus dévoué au sein du travail de groupe, cela permet d'éviter (ou du moins d'atténuer) le phénomène du « free-rider » qui décrit le membre du groupe étant le moins actif dans la réalisation du travail [**Moccozet, Tardy, Opprecht et Michel, 2013**].

2.2.2 Application à la programmation

La programmation est un domaine où la gamification peut s'incruster parfaitement. Un des objectifs de la gamification, pertinent notamment dans le milieu scolaire, est l'engagement social. En effet les jeunes étudiants ont tendances à se démotiver très vite et l'entraide est alors un facteur clé.

Il est important de motiver les élèves pour qu'ils ne décrochent pas et éviter que cela se répercute sur d'autres cours. La gamification apporte ici un aspect de communauté, c'est-à-dire permettre aux étudiants de partager leurs connaissances et de travailler en équipe par exemple [**Lee et Hammer, 2011**]. Il a été prouvé que la participation et le partage des connaissances avaient un impact significatif sur la réussite [**Ahmad, Azmi et Iahad, 2015**].

Beaucoup de plateformes reprennent les codes du jeux-vidéo afin d'enseigner des langages de programmation. « CodeSchool », « RobotCode », « Golden Quest » et d'autres, sont des plateformes qui mettent en avant des barres de progressions, des animations colorées pour attirer le regard ou encore des notions d'évolution au fur et à mesure de la progression, ce qui donne l'envie à l'utilisateur d'aller plus loin et de parcourir des niveaux supplémentaires.

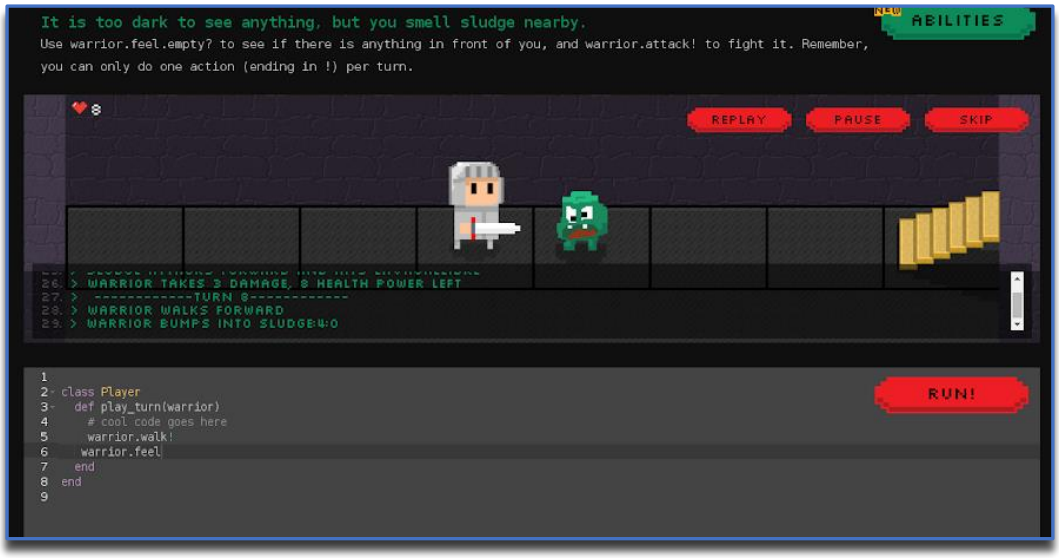


Figure 12 : Exemple de plateforme de jeu pour apprendre la programmation, reproduit depuis Ruby Warrior

La Figure 12 montre le principe de la plateforme « Ruby Warrior » qui permet à ses utilisateurs d'apprendre le langage web Ruby de manière ludique via des aspects de gamification. Via son code, l'utilisateur en apprentissage va pouvoir avancer dans l'aventure afin de progresser au travers de différents tableaux en combattant, récupérant de l'expérience etc. dans l'objectif de finir le jeu.

Cette notion de progression et les différents aspects qui composent la plateforme tend à augmenter l'implication des utilisateurs et ainsi leur apprendre plus facilement le langage.

Pour garantir un usage réussi de la gamification, il est nécessaire d'impliquer l'utilisateur en lui donnant une sensation de progression et d'interaction avec des personnes ayant sensiblement le même but que lui. Cela aura pour effet de multiplier son engagement et sa volonté de réussir [**Knutas, Ikonen, Nikula et Porras, 2014**].

2.3 Solutions existantes

Ce dernier point concerne les solutions existantes, avec d'un côté les solutions d'apprentissage basées sur l'évaluation du code via les tests ou orientées qualité, et de l'autre côté celles qui se focalisent sur la gamification.

2.3.1 Solutions basées sur les tests

2.3.1.1 Mumuki

Mumuki est une plateforme fournissant une évaluation sur base d'un programme. C'est un logiciel utilisé pour guider les étudiants dans l'apprentissage de la programmation qui permet de réaliser des exercices sur une interface et d'avoir ensuite une correction immédiate sur le travail effectué.

Figure 13 : Exemple de retour effectué par l'interface de Mumuki[10]

La Figure 13 reprend un exemple d'utilisation de la plateforme, là où l'utilisateur peut envoyer son code afin d'obtenir un retour sur celui-ci. Ce système se base sur des tests unitaires qui sont fourni par le professeur ayant créé l'exercice. Par ces tests, il va définir le

[10] Retour sur un code Python bien indenté, reproduit depuis https://mumuki.io/home/blog.html.

comportement attendu par le programme et Mumuki va se charger d'exécuter ceux-ci pour attester ou non la fiabilité du programme. Finalement, un retour est fourni aux étudiants leur indiquant la qualité de leur travail.

2.3.1.2 INGInious

INGInious est une plateforme d'évaluation de code d'étudiants créée à l'université de Louvain. Le principe est le suivant : fournir un feedback complet sur un code remis par un élève dans le cadre d'un exercice de programmation. Ainsi, la plateforme va utiliser des tests unitaires afin de tester le code et de s'assurer de la pertinence de celui-ci. Le programme fournit par l'étudiant est donc exécuté avec des valeurs d'entrées déterminées par INGInious pour ensuite vérifier les valeurs de sorties et les éventuelles erreurs.

Le feedback fournit dépend du contexte, il ne sera pas le même pour une erreur récurrente dans un exercice particulier ou pour une erreur qui peut survenir dans tous types d'exercices. Là où un compilateur n'est pas forcément en mesure de différentier ces différents cas, INGInious utilise une vérification statique pour reconnaître les erreurs les plus fréquentes et renvoyer l'information à l'utilisateur [**Derval, Gego, Reinbold, Frantzen et Van Roy, 2015**].

2.3.1.3 Property-Based Testing

Un autre système, proposé par l'université Polytechnique de Madrid, se base sur une approche dite de « property-based testing » pour fournir une évaluation à un programme. Le système est ici semi-automatisé, le programme est évalué via les tests au niveau de sa complexité, mais c'est au professeur qu'il incombe de remettre une note finale.

Pour l'évaluation, l'outil de property-based testing va utiliser « QuickCheck[11] » afin de générer des valeurs aléatoires pour certaines variables. Ainsi, ces variables viendront vérifier une propriété booléenne qui se voudra en théorie vraie. Cette opération va être répétée un certain nombre de fois, si lors de ce processus une valeur « False » apparait ou qu'une erreur vient terminer le programme, l'instance de test se termine [**Benac, Fredlund et Hughes, 2016**].

2.3.2 Solutions orientées qualité

2.3.2.1 SQO-OSS

SQO-OSS est un framework offrant un modèle d'évaluation de programme basé sur des métriques avec une intervention humaine minimale, et s'utilise spécifiquement pour les logiciels libres [**Samoladas, Stamelos, Spinellis, et Gousios, 2008**].

Le processus d'évaluation est divisé en deux phases : la définition du modèle d'évaluation et la définition de la méthode d'agrégation. Dans la première phase, le modèle est défini sur la base de critères de qualité et de métriques associées. Dans la deuxième phase, la méthode d'agrégation représente la collecte de données et l'agrégation des résultats de mesure afin de déterminer la qualité d'un code.

[11] Outil de "Property-based testing" inspiré par Haskell. Il utilise la génération aléatoire de cas de test au lieu de la génération de test orientée vers la couverture.

2.3.3 Solutions basées sur la gamification

Lors de la section 2.2.2, certaines plateformes ludiques pour l'apprentissage de la programmation ont déjà été citées.

Il existe cependant certaines solutions plus théoriques qui méritent d'être étudiées. Il y a par exemple des plateformes d'e-learning reprenant le processus théorique d'un cours à savoir :

(1) Le professeur explique sa matière
(2) L'étudiant effectue un exercice
(3) L'étudiant soumet son exercice à un professeur en vue d'une correction
(4) Le professeur rend son jugement à l'étudiant

Ce processus met en place des aspects de gamification de façon à le rendre le plus ludique possible. Les exercices effectués se font au moyen de « quêtes » comme dans un jeu vidéo. La plateforme permet également une communication et une évaluation entre les élèves, ce qui favorise l'entraide et fait écho à l'évaluation des groupes mentionnées plus haut dans ce document [**Swacha et Baszuro, 2013**].

2.3.3.1 Sleuth

Sleuth est une plateforme développée pour enseigner la programmation à des étudiants en utilisant de la gamification tout au long du processus. Le but est d'acquérir certains concepts au niveau de la syntaxique et de la sémantique concernant un ou plusieurs langages.

La plateforme se veut comme un jeu d'enquêtes où un étudiant incarne un détective qui doit corriger des codes défectueux pour progresser et développer les niveaux suivants. L'élève soumet ainsi le code corrigé et reçoit un feedback du chef de la brigade -le système- qui va lui indiquer ses conclusions sur le code. Dans ce retour se trouvent des erreurs tant syntaxiques que sémantiques. Le but ici est que l'étudiant répète l'exercice jusqu'à arriver à une solution parfaite. De cette façon, il débloque le niveau supérieur et peut progresser dans le jeu et dans son apprentissage [**Katan et Anstead, 2020**].

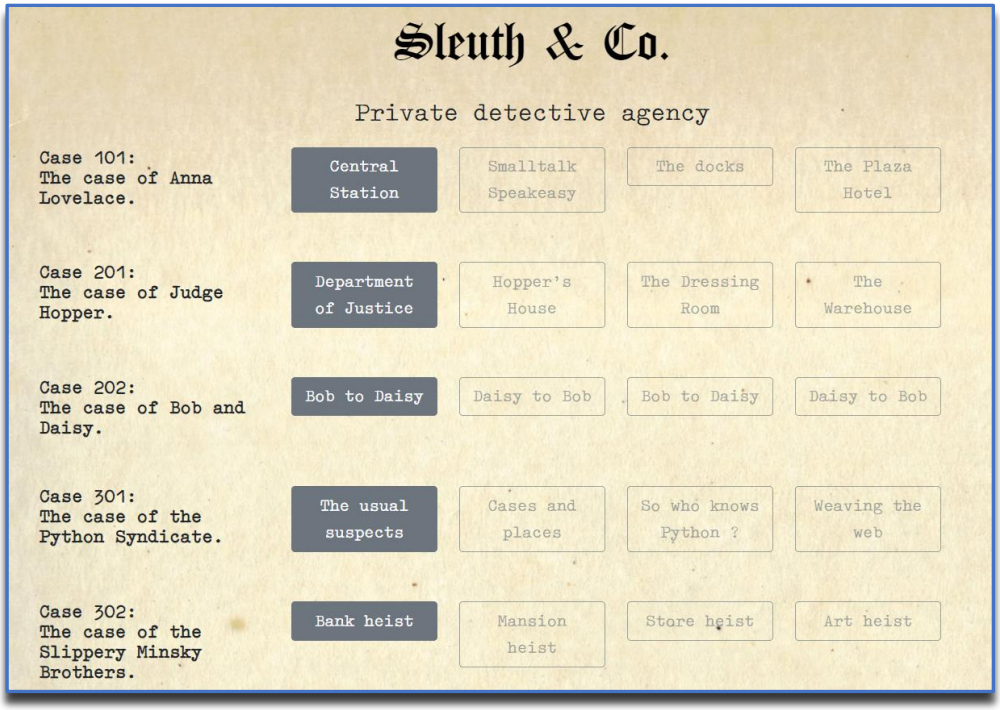


Figure 14 : Interface de la plateforme Sleuth[12]

La Figure 14 représente l'interface principale de la plateforme, c'est ici que l'étudiant peut résoudre des exercices qui sont matérialisés sous forme d'enquêtes. Le système montre ainsi les niveaux à débloquer et ceux que l'étudiant a déjà résolu.

C'est là un exemple parfait de l'utilisation de la gamification dans un programme d'apprentissage. Le système propose des tâches qui sont répétitives (bien que quelques variantes d'exercices puissent être générées par le système), mais ludiques, dans une interface qui soulève chez l'étudiant l'envie d'avancer et de finir le « jeu ». Cela le motive à persévérer dans son apprentissage et à éviter son désintérêt pour la matière.

2.3.3.2 Codewars

Codewars[13] est une plateforme qui permet à ses utilisateurs d'apprendre et d'approfondir leurs compétences en programmation via la gamification. Elle propose un large éventail d'exercices à résoudre pour améliorer la logique de programmation des utilisateurs, et ce dans plusieurs langages.

De plus, afin de donner l'envie aux utilisateurs de continuer à progresser, elle propose un profil utilisateur personnalisable avec des statistiques et des récompenses en fonction des exercices faits, ainsi qu'une barre de progression et un classement global.

2.4 Résumé

Ce chapitre reprend toute la revue de littérature qui a été faite pour ce mémoire. Il permet d'explorer différents thèmes en lien avec la question de recherche. Dans un premier temps, l'analyse sur l'évaluation en milieu scolaire montre comment fournir une cotation et un retour

[12] Menu principal de la plateforme Sleuth, repris depuis http://doc.gold.ac.uk/www/340/.
[13] Plateforme Codewars : https://www.codewars.com/dashboard.

pertinent à un étudiant, lui permettant ainsi d’améliorer sa compréhension de la matière. Cette évaluation peut prendre plusieurs formes (formative, sommative...), possédant chacune leurs propres avantages.

Ensuite, il y a plusieurs façons d’évaluer un programme, deux des plus utilisées d’entre elles sont abordées ici : l’évaluation basée sur les tests d’un côté, et orientée qualité, basée sur les normes ISO de l’autre. Se baser sur les tests permet d’évaluer des résultats pour des exercices précis, à condition que les tests soient prédéfinis. L’évaluation orientée qualité permet un processus automatisé applicable pour tout code, à condition de définir des mesures précises pour évaluer au mieux la qualité globale d’un programme.

La gamification quant à elle offre un moyen de captiver l’élève afin qu’il se sente plus impliqué dans la tâche qu’il réalise. Il y a aujourd’hui un rapprochement important entre la gamification et le monde de l’apprentissage en général. Tout comme il est important de pointer du doigt les faiblesses d’un étudiant afin qu’il puisse s’améliorer, il l’est tout autant de mettre en évidence ses succès pour booster sa confiance en lui. Cela peut se faire en le récompensant pour ses réussites et objectifs atteints.

La dernière partie de ce chapitre repose sur une analyse des solutions déjà existantes tant au niveau de l’évaluation de programmes que de la gamification. Cela permet de constater que c’est un sujet recherché, mais il n’y a pas encore de solutions parfaites ; chaque système a ses qualités comme ses défauts et il est difficile d’arriver à une évaluation automatique précise, pertinente et robuste.

3 Conception

Le choix de conception s'est porté sur la mise en place d'une plateforme indépendante, tout en gardant à l'esprit une possible intégration dans le futur. Cela permet de gagner du temps notamment lors de l'implémentation, en évitant de devoir installer tous les éléments nécessaires pour lancer le système existant, et de lire et comprendre l'entièreté du code pour pouvoir le modifier par la suite. De plus, il n'a pas été jugé utile de le faire car la solution apportée par ce mémoire est complémentaire au chatbot précédemment développé, les deux systèmes se suffisent à eux-mêmes.

Actuellement, un élève de bloc 1 à l'UNamur dispose de deux outils pour parfaire son apprentissage du cours « *INFOB131 Introduction à la programmation* » :

(1) **Webcampus**, lui permettant de récupérer les cours théoriques, les énoncés des exercices et d'avoir une interaction minime avec les professeurs

(2) **CHATssistant**, le projet mis en place lors de l'année académique 2019-2020 et qui consiste en un chatbot qui va servir de support à l'étudiant en difficulté lors de la réalisation de ses exercices

À travers ce chapitre est détaillée point par point la solution qui a été mise en place. Par-là, le cadre global du travail effectué, la description de la solution et des différents composants mis en place, avec notamment la description des mesures et des algorithmes d'évaluation et de correction de lignes erronées, et pour finir, la description de son implémentation.

3.1 Cadre de travail

Cette section décrit la solution à mettre en place déterminée sur base des recherches effectuées dans l'état de l'art. L'analyse qui en ressort permet de faire les choix d'implémentation adéquats dans le cadre du projet.

3.1.1 Modèle d'évaluation proposé

Compte tenu des sujets étudiés dans le chapitre précédent, il est possible de déduire certains éléments sur la manière d'apporter une cotation pertinente à un programme informatique. La section 2.1.4 montre que le modèle systémique encourage à voir plus loin que l'étudiant pour comprendre ses difficultés, mais que trop s'adapter à un élève spécifique fournit un enseignement différent pour tous. La mise en place d'un système « rigide » peut faire peur sur le papier, mais il garantit une formation égalitaire entre tous les élèves et des retours sur la matière, identiques pour tous les individus. C'est de cette façon qu'ils obtiendront le retour le plus formel possible sur leur travail. La plateforme s'inscrit donc dans un plan plus formatif, permettant aux élèves de s'évaluer tout au long de l'année et aux professeurs de voir leur avancée. À terme, la plateforme pourrait potentiellement être utilisée pour des évaluations sommatives si les enseignants l'estiment assez pertinente et stable pour ce faire.

Dans le cadre du cours « *INFOB131 Introduction à la programmation* », les étudiants sont amenés à faire des travaux individuels, mais aussi des travaux de groupes, qui donnent généralement des programmes plus conséquents. La plateforme doit donc être capable de s'adapter aux codes soumis de façon à pouvoir évaluer des travaux de toutes tailles. Comme l'analyse se fait sur un programme fournit par un utilisateur connecté au système, il n'est pas possible actuellement de donner une note à différents membres d'un groupe. Dans tous les cas,

malgré les observations faites au point 2.1.3, la note serait la même pour tous sachant que le système ne peut tenir compte de l'investissement de chaque étudiant dans un travail, l'évaluation étant entièrement basée sur le code. Pour finir, la plateforme est dédiée à Python, étant le langage de programmation utilisé dans le cours ciblé. À terme, il pourrait être intéressant d'élargir le panel de langages pris en compte par le système.

3.1.2 Évaluation du code

La solution proposée ici opte pour une évaluation du code source axée sur la qualité (plutôt que sur les tests), car l'évaluation et le feedback peuvent être très précis, sans devoir définir des exercices et des tests au préalable.

Pour ce faire, la norme ISO/IEC 25000 est utilisée, tant elle est la référence pour évaluer la qualité d'un logiciel à l'heure actuelle. Aussi, à l'aide du modèle de mesure de l'information proposé par **Abran, Al-Qutaish, Desharnais et Habra (2005)**, des mesures sont définies pour chaque critère de qualité défini par l'ISO 25010.

Le but de cette recherche est d'évaluer les programmes des étudiants débutant dans le domaine de la programmation. Il est important de cibler les erreurs à vérifier et les points à évaluer afin de ne pas être trop punitif et garder à l'esprit l'aspect pédagogique. Dans ce sens, les mesures définies doivent correspondre à des éléments importants notamment au niveau des bonnes pratiques et de la bonne fonctionnalité du code. Il peut dès lors être intéressant de mettre un ordre d'importance aux différentes mesures. Aussi, il faudra différencier les petits scripts des programmes plus importants, en fonction des éléments présents dans le code. Cela peut se faire en appliquant certaines mesures dynamiquement en fonction du code reçu et de son contenu.

Dernièrement, les codes à évaluer appartenant majoritairement à des étudiants débutants, ils sont souvent truffés d'erreurs syntaxiques parfois plus graves, empêchant la compilation et diverses analyses. Le but est donc également de définir un moyen de passer outre ces lignes qui posent problèmes afin de pouvoir procéder à l'analyse et donner un retour pertinent à l'élève dans tous les cas.

3.1.3 Gamification du système

Un des objectifs est de proposer à l'étudiant un retour clair et compréhensible sur son travail. L'usage de la gamification est ici un plus afin de le motiver à pratiquer des exercices. L'idée est de fournir une plateforme ludique, avec différents aspects qui vont lui donner un intérêt à s'entrainer d'avantage ainsi que des objectifs à atteindre pour qu'il puisse se voir progresser. Aussi, la mise en relation avec d'autres élèves du même cursus permet d'apporter un peu de compétitivité, mais aussi de l'entraide afin qu'ils se tirent vers le haut. C'est pourquoi il est intéressant d'incorporer ces divers éléments de gamification dans la solution.

3.2 Description de la solution

Dans ce point est décrite la solution mise en place, les exigences auxquelles elle répond, les cas d'utilisation, l'architecture du système ainsi que les divers outils et technologies utilisés. La méthode de construction d'une mesure ainsi que l'algorithme de correction de lignes sont également détaillés ici. Pour terminer, la possibilité d'une intégration avec un système existant est évoquée.

3.2.1 Exigences fonctionnelles

Pour atteindre ses objectifs, la solution mise en place doit satisfaire différentes exigences fonctionnelles :

(1) La possibilité d'évaluer un code dans le cadre d'un exercice libre, un devoir, un projet... dans le but d'obtenir un retour complet sur la qualité du programme en question. La qualité du programme est évaluée selon les différents critères proposés par l'ISO 25000, en fonction des mesures définies en section 3.3 pour chacun d'eux. L'évaluation de la qualité du programme doit également pouvoir se faire si le code envoyé n'est pas compilable et/ou ne respecte pas la syntaxe de Python.

(2) Le système doit retenir les différentes notes reçues par les étudiants ainsi que les diverses informations relatives aux évaluations faites, afin de pouvoir montrer la courbe de progression de chacun et de pouvoir faire et montrer des statistiques relatives à cela telles que la moyenne d'un élève sur une semaine ou sur l'année, ses moyennes par critère, le classement hebdomadaire...

(3) L'étudiant doit pouvoir débloquer des récompenses pour avoir relevé des défis afin de pouvoir personnaliser son profil et de lui donner un but supplémentaire pour s'exercer. De l'autre côté, un professeur doit pouvoir ajouter, modifier et supprimer des récompenses en les associant à des défis.

3.2.2 Exigences non-fonctionnelles

La plateforme répond aussi à un ensemble de critères non-fonctionnels :

(1) **Exigences d'apparence et ressentit** : la plateforme se veut visuellement attractive pour les utilisateurs. Les étudiants comme les professeurs doivent se sentir à l'aise et passer un bon moment pendant l'utilisation du système. L'apparence de la plateforme avec les aspects de gamification doivent inciter notamment les élèves à la réutiliser par la suite.

(2) **Exigences d'accessibilité** : l'interface se veut suffisamment intuitive pour que l'utilisateur puisse facilement s'y retrouver afin de lui garantir une facilité d'utilisation constante.

(3) **Exigences de performance/de disponibilité** : les utilisateurs peuvent accéder au système sans discontinuité, afin de garantir une disponibilité constante à l'utilisateur.

(4) **Exigences de maintenabilité** : le système est conçu de façon à pouvoir intégrer de nouvelles modifications relativement facilement. Ce qui veut dire que le système est assez modulable et permet la modification/l'ajout/la suppression d'éléments de manière simple, mais aussi que le code est documenté afin de faciliter la compréhension du système pour quiconque doit reprendre le code par la suite.

(5) **Exigences de sécurité** : le système garantit la confidentialité des données personnelles de l'utilisateur. Aussi, un utilisateur n'a accès qu'à son propre compte. Un étudiant ne peut pas utiliser les fonctionnalités et les données accessibles aux professeurs.

3.2.3 Cas d'utilisation

La plateforme METAssistant est mise en place principalement pour les étudiants à destination du cours « *INFOB131 Introduction à la programmation* ». Mais il est toujours

intéressant d'élargir un maximum le champ d'application d'un système afin de prévoir une éventuelle expansion par la suite. Il doit donc correspondre pour tout élève souhaitant tester la qualité de ses programmes.

Il y a dès lors deux types d'acteurs qui vont pouvoir interagir avec le programme, les étudiants et les professeurs, qui utilisent le système à des fins différentes.

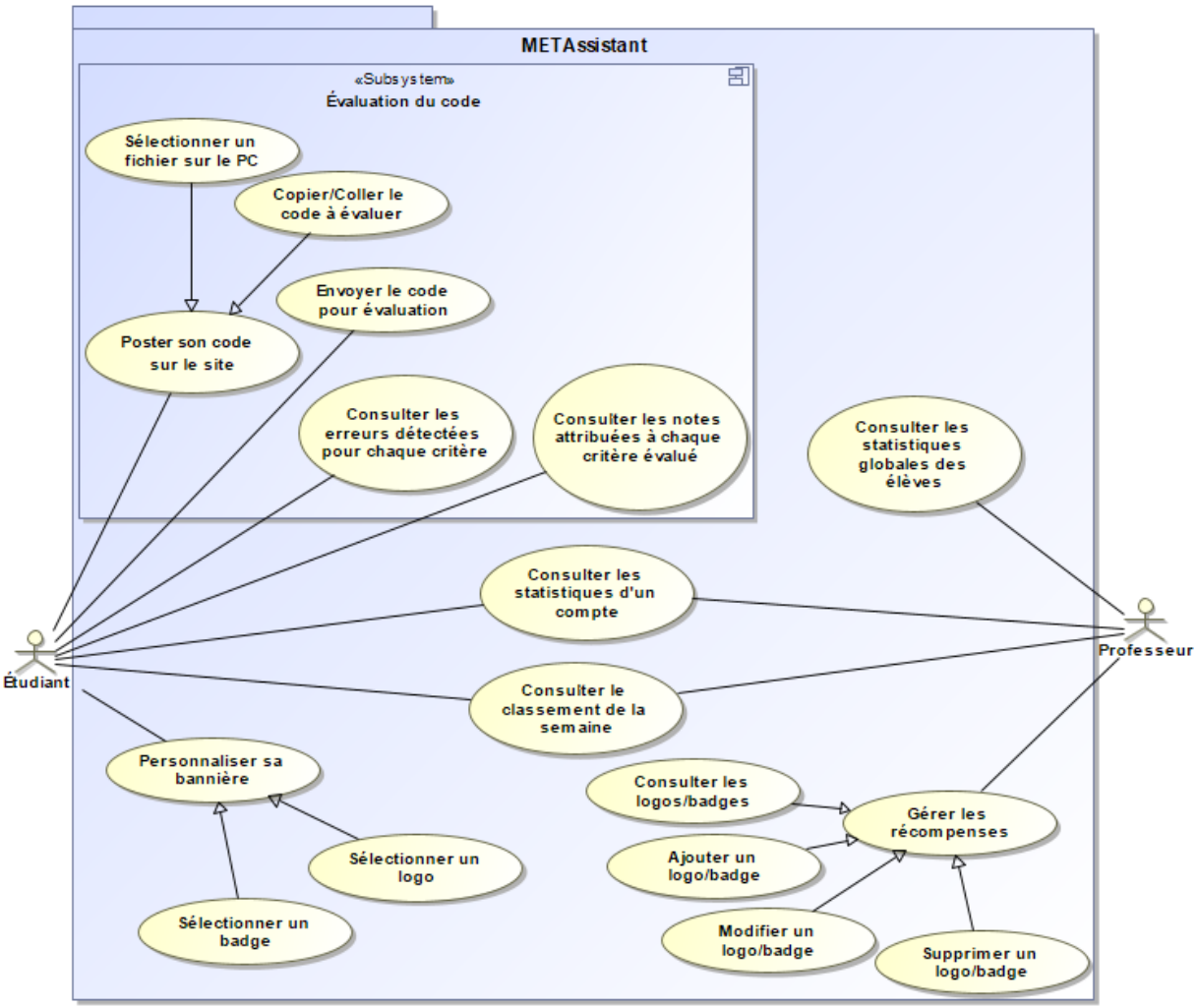


Figure 15 : Diagramme de Use case représentant les différentes fonctionnalités du système

La Figure 15 montre les différentes fonctionnalités offertes par le système. La connexion au système se fait à l'aide des identifiants de l'UNamur. Il est nécessaire pour chaque acteur d'être connecté au système pour pouvoir accéder aux fonctionnalités.

3.2.3.1 Poster son code sur le site

Un étudiant doit pouvoir poster son code sur le site. Cela peut se faire de deux manières, (1) en faisant un copier/coller de son code dans une zone appropriée, ou (2) en sélectionnant directement un fichier via l'explorateur de fichiers.

3.2.3.2 Envoyer le code pour évaluation

Un étudiant doit pouvoir envoyer son code afin de récupérer le détail de l'évaluation.

Précondition : Il est nécessaire d'avoir effectué le 'use case' décrit en 3.2.3.1 pour pouvoir utiliser celui-ci.

Postcondition : Les détails de l'évaluation sont affichés à l'utilisateur.

3.2.3.3 Consulter les erreurs détectées pour chaque critère

Un étudiant doit pouvoir consulter les diverses erreurs détectées dans son code.

Précondition : Il est nécessaire d'avoir effectué le 'use case' décrit en 3.2.3.2 pour pouvoir utiliser celui-ci.

3.2.3.4 Consulter les notes attribuées à chaque critère évalué

Un étudiant doit pouvoir consulter les notes attribuées aux différentes mesures.

Précondition : Il est nécessaire d'avoir effectué le 'use case' décrit en 3.2.3.2 pour pouvoir utiliser celui-ci.

3.2.3.5 Consulter les statistiques d'un compte

Un étudiant peut accéder à son profil et voir ses statistiques. Il doit également pouvoir, comme un professeur, visiter le profil d'un autre étudiant et donc ses statistiques.

3.2.3.6 Consulter le classement de la semaine

Un étudiant, comme un professeur, doit pouvoir accéder au classement de la semaine.

3.2.3.7 Personnaliser sa bannière

Un étudiant doit pouvoir personnaliser sa bannière de deux manières, (1) en choisissant un logo parmi ceux qu'il possède, (2) en choisissant un badge parmi ceux qu'il possède.

3.2.3.8 Consulter les statistiques globales des élèves

Un professeur doit avoir accès à des statistiques plus générales au niveau des étudiants afin d'avoir un aperçu sur le niveau global des élèves et de pouvoir s'y adapter et potentiellement adapter son cours.

3.2.3.9 Gérer les récompenses

Un professeur doit pouvoir gérer les récompenses telles que les badges et logos que les élèves vont recevoir en accomplissant des défis pour les débloquer. Le professeur peut effectuer quatre opérations différentes. (1) Créer une récompense en définissant un objectif à atteindre pour la débloquer, (2) voir la liste des récompenses qui sont déjà implantées, (3) modifier les récompenses existantes, et (4) les supprimer si nécessaire.

3.2.4 Architecture du système

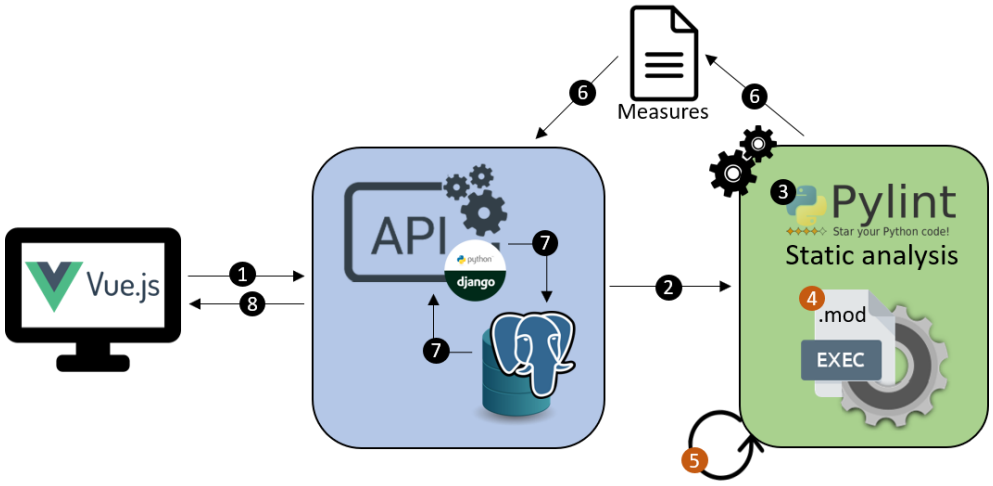


Figure 16 : Architecture du système et le chemin d'exécution pour l'évaluation d'un programme

La Figure 16 représente une vue de base de l'architecture de la plateforme. Elle peut être divisée en trois blocs : l'interface utilisateur, l'API et l'algorithme d'évaluation. Premièrement, l'interface utilisateur (symbole d'écran d'ordinateur) est écrite en Vue.js et intègre des aspects de gamification comme décrit dans la section 3.4.3. L'algorithme d'évaluation (bloc vert) est expliqué plus en détail dans la suite de la section. Ensuite, l'API et l'accès aux données (bloc bleu) permettent à un utilisateur de récupérer et de mettre à jour les données de la base de données PostgreSQL, d'évaluer son programme et d'obtenir un retour. L'API est implémentée avec le framework Django et suit les conventions REST. Le chemin d'exécution de l'évaluation est également visible, avec des étapes numérotées de 1 à 8, depuis le moment où le code est soumis jusqu'à la récupération du feedback. À noter que les éléments numérotés en orange définissent une exécution conditionnelle qui n'est utilisée que lorsqu'une erreur se produit dans le processus d'évaluation.

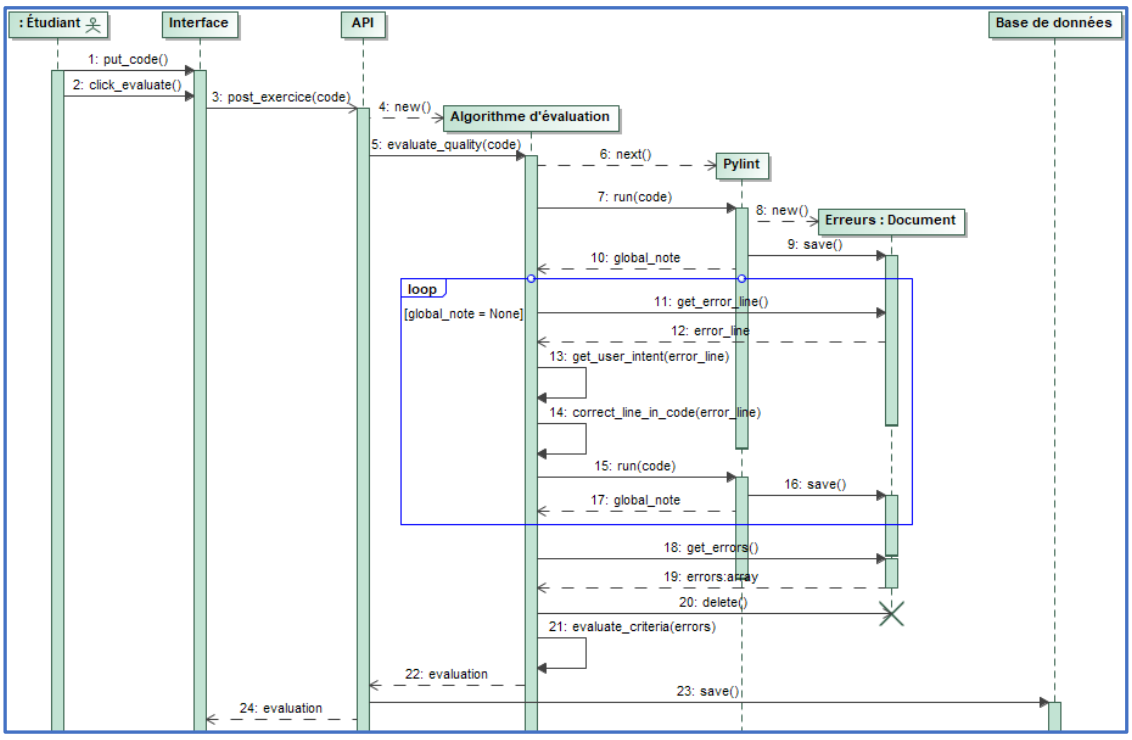


Figure 17 : Diagramme d'interactions de l'évaluation d'un programme par un étudiant

La Figure 17 représente le diagramme séquence de l'évaluation d'un programme, qui décrit les différentes interactions entre les composants, de l'envoi du code à l'affichage des résultats. Par soucis de concision et de clarté, le cas décrit par ce schéma ne prend pas en compte les éventuelles erreurs empêchant le bon fonctionnement du système.

Tout d'abord, l'étudiant met son code sur la plateforme et clique sur « évaluer ». Une requête est envoyée à l'API qui va s'occuper de la suite. Le code est envoyé à l'algorithme d'évaluation qui va lancer une analyse de code statique à l'aide de la librairie Python "Pylint" afin de récupérer les codes d'erreur pertinents pour l'évaluation du programme de l'utilisateur. Si une erreur se produit [c.-à-d. que l'outil d'analyse a rencontré une erreur fatale (dans ce cas la note renvoyée n'a pas de valeur assignée, elle vaut « None »)], elle est enregistrée dans un fichier et le numéro de ligne de l'erreur est renvoyé et utilisé (1) pour récupérer l'intention de l'utilisateur, c'est-à-dire l'action que l'utilisateur voulait effectuer selon l'algorithme, et (2) afin de corriger la ligne erronée en fonction des éléments qui la composent et de l'intention détectée. L'algorithme de correction de ligne (ACL) est décrit plus en détails à la section 3.2.7. Une fois la ligne corrigée, une autre analyse est faite et s'il y a une erreur, elle est renvoyée à l'ACL, et ce en boucle jusqu'à correction de toutes les erreurs fatales. Quand il n'y en a plus, l'analyse s'exécute et renvoie les erreurs détectées relatives au code et les enregistre dans le fichier. Sur base de ces erreurs, les mesures concernées vont pouvoir être évaluées et le résultat est renvoyé d'abord à l'API qui va enregistrer les données en base de données, puis à l'interface pour pouvoir l'afficher à l'utilisateur. À noter que le calcul de la moyenne générale pour un code est une moyenne pondérée des notes des mesures par rapport à leur importance.

3.2.5 Technologies et outils utilisés

Cette section décrit les technologies et les outils utilisés ainsi que les raisons qui ont amené ces choix. La manière de construire une mesure et l'algorithme de correction de lignes sont également expliqués. La possible intégration du système avec l'existant est aussi abordée.

3.2.5.1 Vue.js

Dans le cadre de ce projet, Vue.js a été utilisé pour mettre en place le front-end, c'est-à-dire l'interface avec laquelle les utilisateurs vont interagir. Vue.js est un des frameworks les plus utilisés dans le développement web[14]. C'est un framework Javascript open-source, offrant une documentation bien structurée et complète[15] et qui est facile à prendre en mains. Comme React ou Angular, c'est un framework orienté composants qui permet d'étendre les éléments HTML en encapsulant le code dans des composants réutilisables via des balises HTML. À noter que ces trois frameworks se valent au niveau des performances[16]. Il permet de créer facilement des interfaces utilisateurs à l'aide des langages HTML, CSS et Javascript, contrairement à React qui se base énormément sur le Javascript, même pour la structure HTML avec JSX et la gestion du CSS, ce qui complexifie son utilisation, et Angular qui se base sur du TypeScript. Une autre force de ce framework réside dans son coté « progressif », qui permet d'intégrer Vue.js à tout un ensemble de technologies (comme c'est le cas dans ce projet avec 'Django' notamment). Il facilite également le routage entre les composants et met à jour automatiquement la vue quand

[14] Statista (2021) https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/.
[15] Vue.js documentation https://vuejs.org/v2/guide/.
[16] Benchmark des frameworks web https://www.stefankrause.net/js-frameworks-benchmark8/table.html.

les données changent grâce au data-binding. L'utilisation de ce framework permet de gagner du temps dans le développement car il permet de déléguer la gestion de certains aspects, il est facile à prendre en main et bien documenté, performant et adaptable.

3.2.5.2 Django

Django a été utilisé pour mettre en place l'API, c'est-à-dire la couche intermédiaire entre la vue et la base de données. Django est un framework Python pour le web développé en 2003 et il fait partie des frameworks back-end[17] les plus utilisés à ce jour[18]. Il permet une mise en place et un développement rapide du projet, et dispose d'une librairie permettant de mettre en place une API respectant les contraintes REST. À nouveau, c'est un framework open-source et bien documenté, proposant des guides complets pour la mise en place du système. Django repose sur une structure MVT (Model-View-Template), inspirée du modèle MVC (Model-View-Controler) classique, qui propose un couplage faible en séparant les responsabilités de chaque module, et permet d'établir une structure très claire du projet dès le début du développement. Django peut surtout s'avérer utile car il est capable de gérer tout un ensemble d'interactions seul, via un ORM (Object Relationnal Mapping). Il permet aux développeurs de ne pas se soucier des requêtes SQL pour manipuler les données, cet aspect est géré implicitement par le framework. Cela permet au développement de se concentrer sur le code dit « métier », c'est-à-dire celui qui est propre au domaine d'application. Il offre également une interface d'administration générée automatiquement très utile pour gérer la base de données pendant le développement sans devoir passer par un tiers. Aussi, il s'intègre très facilement avec Vue.js et beaucoup de guides sont disponibles. De plus, sachant que cette technologie a été utilisée pour la précédente recherche CHATssistant, et en prévention d'une éventuelle intégration d'un système vers l'autre, il est intéressant de garder ce framework pour la faciliter.

3.2.5.3 PostgreSQL

PostgreSQL est le système de gestion de base de données (SGBD) qui a été choisi pour la réalisation de ce projet. Le système du METAssistant repose en partie sur la manipulation de données qui entourent l'utilisateur (sa moyenne pour un critère, son nombre de tentatives, de récompenses...), ces données doivent donc être conservées au sein d'une base de données fiable et rapide d'accès.

PostgreSQL s'avère donc particulièrement utile dans la mise en place de la plateforme car il permet de gérer implicitement des aspects de notre système. La gestion de l'utilisateur et de ses droits d'accès est ainsi automatisée, de même que certaines notions de sécurité qui sont déployées automatiquement avec le système. De plus le framework Django utilisé dans le cadre de ce travail permet une intégration relativement simple d'une base de données de type PostgreSQL, ce qui s'avère très intéressant afin de gagner du temps dans le cadre de l'implémentation.

Comme pour les technologies citées aux points précédents, le système CHATssistant se repose aussi sur une base de données PostgreSQL, il était donc cohérent de recourir à un SGBD de ce type (en plus des avantages cités ci-avant) afin de garantir une continuité et de faciliter une intégration future entre les deux projets.

[17] Back-end : tout ce qui n'est pas directement accessible à l'utilisateur, la partie immergée de l'iceberg.
[18] Statistics & Data (2021) https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/.

En ce qui concerne les images à sauvegarder en base de données, un champ est prévu afin d'enregistrer une chaine de caractères, de quoi accueillir une URL. Dès lors, toute image utilisée dans la plateforme est enregistrée dans Cloudinary[19] pour faciliter la réutilisation, avoir des URLs fixes, d'une source sûre et contrôlée, et donc éviter tout problème d'intégrité au sein de la plateforme.

3.2.5.4 Pylint

Pylint est un outil d'analyse statique de code pour Python qui vérifie cinq types d'erreurs :

(1) 'convention' pour les violations des standards de programmation,
(2) 'warning' pour les problèmes spécifiques à Python,
(3) 'error' pour les éléments qui sont probablement des bugs,
(4) 'refactor' pour les 'code smells' (c.-à-d. les mauvaises pratiques),
(5) 'fatal' pour les erreurs empêchant l'exécution.

Contrairement à des outils comme Flake8, Pylint est hautement configurable et des erreurs spécifiques peuvent être activées/désactivées si nécessaire. Il existe également une bibliothèque permettant d'invoquer Pylint de manière programmatique[20]. De plus, Pylint propose un grand nombre de checkers et de codes erreur avec plus de 280 types d'erreurs différents contre une cinquantaine pour Flake8, ce qui permet d'avoir un rapport très détaillé sur les erreurs rencontrées. De plus, la documentation de Pylint est également très bien détaillée. Tout cela fait de lui un excellent outil pour vérifier les bonnes pratiques dans le code et pour évaluer efficacement les mesures syntaxiques, notamment en matière de fonctionnalité, de maintenabilité et de sécurité.

La configuration de Pylint peut se faire de deux manières, (1) via les rcfiles ou (2) directement dans la commande. Les rcfiles sont des fichiers de configuration qui peuvent être générés via l'option '--generate-rcfile' dans la commande Pylint. Ce fichier contient diverses options permettant de paramétrer l'analyse. La Figure 18 montre un échantillon de ces paramètres.

```
419   # Maximum number of characters on a single line.
420   max-line-length=80
421
422   # Regexp for a line that is allowed to be longer than the limit.
423   ignore-long-lines=(?x)(
424     ^\s*(\#\ )?<?https?://\S+>?$|
425     ^\s*(from\s+\S+\s+)?import\s+.+$)
426
427   # Allow the body of an if to be on the same line as the test if there is no
428   # else.
429   single-line-if-stmt=yes
```

Figure 18 : Exemple de paramètres disponibles dans un rcfile de Pylint

Outre les rcfiles, il est possible de modifier les paramètres directement dans la commande Pylint, en spécifiant le nom du paramètre et la valeur à assigner. La librairie Python 'Pylint' fournit la méthode 'Run()' qui permet d'exécuter Pylint via un programme.

[19] Cloudinary : service qui permet de gérer des images et des vidéos dans le cloud.
[20] Pylint, exécution et invocation http://pylint.pycqa.org/en/latest/user_guide/run.html.

```
result = Run(
    [input_file,
    '--errors-only',
    '--evaluation='+evaluation_method,
    '--disable=all',
    '--enable='+enabled_checks],
    reporter=TextReporter(output_file),
    exit=False)
```

Figure 19 : Exemple d'utilisation de la méthode 'Run()' de la librairie Python 'Pylint'

La Figure 19 reprend une utilisation de la méthode 'Run()' où :

(1) 'input_file' est le fichier '.py' contenant le code de l'étudiant.
(2) '--errors-only' est une option spécifiant que le rapport final de Pylint ne doit contenir que les erreurs détectées. Si cette option n'est pas mise, le programme renvoie un rapport détaillé contenant des statistiques sur le code en question.
(3) '--evaluation' est une option permettant de redéfinir la méthode d'évaluation de Pylint
(4) '--disable' est une option permettant de désactiver les checkers spécifiés après le '='. La valeur 'all' permet de désactiver la totalité des checkers. C'est utile quand il est question d'activer certains checks en fonction du code afin de paramétrer aisément les erreurs à renvoyer.
(5) '--enable' est une option permettant d'activer les checkers spécifiés après le '='. 'enabled_checks' est une liste de noms de checkers séparés par des virgules. Comme l'option '—disable=all' est mise, seuls les checkers spécifiés ici seront pris en compte.
(6) Les deux derniers arguments permettent respectivement (a) de spécifier un endroit pour enregistrer les résultats dans un fichier texte, et (b) de continuer l'exécution du programme après l'exécution de Pylint.

Le type d'erreur 'fatal' que Pylint vérifie correspond à des erreurs bloquantes pour le processus. Ce sont les erreurs fatales qui sont abordées dans la section 3.2.4. Quand ce type d'erreur est détecté, l'analyse s'arrête et renvoie uniquement la ligne concernée. Dès lors, il faut pouvoir corriger cette ligne afin de pouvoir continuer l'analyse et avoir le retour attendu concernant les erreurs détectées dans le code par Pylint. Ici, cette correction se fait grâce à l'algorithme décrit en section 3.2.7.

3.2.5.5 Distance de Damerau-Levenshtein

La distance de Damerau-Levenshtein calcule la distance d'édition entre deux chaînes de caractères. Il s'agit d'une extension de la distance de Levenshtein, elle est utile notamment pour la détection de typos. En plus de compter le nombre minimum d'opérations nécessaires pour transformer une chaîne de caractères en une autre, une opération étant une insertion, une suppression ou une substitution, elle prend également en compte la transposition de deux caractères, c'est-à-dire un échange entre deux lettres [**Christanti et Naga, 2018**]. Une version normalisée de D-L permet de calculer le rapport entre la distance et la longueur de la chaîne la plus longue, ce qui donne un meilleur indicateur au niveau de la comparaison des deux chaines. Le résultat appartient à l'ensemble [0,1], où 0 signifie qu'aucune différence n'a été trouvée et 1 que les deux mots sont opposés, sans similarité.

Tableau 3 : Résultats de la comparaison de deux valeurs selon différents algorithmes

Algorithme	Valeur 1	Valeur 2	Résultat
Levenshtein	isinstance	isnistance	2
	from	form	2
Damerau-Levenshtein	isinstance	isnistance	1
	from	form	1
D-L normalisé	isinstance	isnistance	0.1
	from	form	0.25

Le Tableau 3 montre les résultats de ces algorithmes sur deux échantillons où une transposition s'est produite. La distance de Levenshtein compte deux opérations entre les chaines comparés (suppression + insertion), tandis que la distance D-L compte la transposition comme une seule opération. La distance D-L est donc plus pertinente ici. De plus, la distance D-L normalisée tient compte de la longueur et donne une indication plus claire sur le taux de similarité des mots. Elle peut être interprétée comme une marge d'erreur pour la comparaison de chaines. La solution proposée dans ce papier utilise une bibliothèque Python[21] pour calculer la distance d'édition D-L normalisée afin de comparer les éléments d'une ligne d'erreur donnée aux symboles et aux mots-clés de Python. Sur base du résultat, le mot vérifié peut être considéré comme pertinent ou non.

3.2.6 Construction d'une mesure

Le processus d'évaluation de la qualité utilisé ici suit le modèle d'information sur les mesures expliqué dans la section 2.1.6. L'objectif est de définir des mesures pertinentes par rapport au public cible pour évaluer les critères définis par la norme ISO/IEC 25000. Dans un premier temps, les mesures de base sont collectées à partir des attributs du code source. Un exemple de mesure de base est (M1) le nombre de lignes de code ou encore (M2) le nombre d'erreurs de l'analyseur statique. Les mesures de base sont des chiffres bruts recueillis à partir du code, mais elles ne fournissent pas beaucoup d'informations sur la qualité en elle-même. Par conséquent, avec une fonction prédéfinie, les mesures de base sont transformées en mesures plus pertinentes. Par exemple, il peut s'agir du nombre d'erreurs par ligne de code, obtenu en calculant le ratio de 'M2' par 'M1'. Ensuite, cette mesure dérivée doit être associée au critère correspondant. Lors de l'évaluation, la valeur obtenue par la mesure d'une information est interprétée à l'aide d'indicateurs pour attribuer une note à la mesure concernée. Le nombre de mesures augmente la précision de l'évaluation, car chaque mesure correspond à un aspect spécifique du code source.

3.2.7 Algorithme de correction de lignes (ACL)

Pylint étant un analyseur de code statique classique, il manque de flexibilité et peut facilement être stoppé dans son exécution. Considérons un code source où le processus de Pylint aboutit à une erreur fatale à une ligne donnée. Le but de l'ACL est d'utiliser la syntaxe de Python

[21] Librairie Python 'pyxDamerauLevenshtein' (2021) https://pypi.org/project/pyxDamerauLevenshtein.

et le ‘pattern matching[22]’ pour comprendre l'intention de l'utilisateur, et d’essayer de remanier la ligne d'erreur pour qu'elle corresponde à cette intention. La première étape consiste à redéfinir la syntaxe de Python avec (1) tous les symboles et mots-clés réservés du langage et (2) des expressions régulières qui correspondent aux éléments composants la syntaxe de Python (c.-à-d. les expressions, les instructions, les définitions, etc.). Une fois la syntaxe redéfinie, elle devient utile pour comparer et faire correspondre les éléments de la ligne d'erreur de l'utilisateur. Pour ce faire, il est nécessaire de construire l'expression régulière (regex)[23] de la ligne d'erreur en considérant cette syntaxe.

Pour chaque élément de la ligne, l’algorithme essaye d’établir un lien avec un élément de la syntaxe. Premièrement, il vérifie si c’est un symbole, notamment à l’aide de la distance Damerau-Levenshtein comme expliqué dans la section 3.2.5.5. Deuxièmement, si ce n’est pas un symbole, l’algorithme essaye de le faire correspondre avec des éléments basiques de la syntaxe comme une variable ou une expression spécifique. Lorsque l'expression régulière de la ligne d'erreur est construite, l’algorithme essaie de le faire correspondre avec un regex de la syntaxe Python en considérant chaque composant du regex de la ligne d'erreur et en prenant en compte leur ordre d’apparition, le nombre d’éléments dans chacun ainsi que le nombre d’éléments en commun. Pour finir, une liste triée des meilleures correspondances est retournée et est utilisée pour remanier la ligne d'erreur. Le remaniement est effectué en réorganisant les composants dans le bon ordre en tenant compte des patrons correspondants et en corrigeant les éléments erronés. Un exemple d’application de l’ACL est proposé au point 4.1.1.

3.2.8 Intégration potentielle

Le système se veut complémentaire au projet CHATssistant, ce qui impose des contraintes supplémentaires à l’implémentation de la plateforme afin de faciliter la potentielle intégration d’un système dans l’autre. En effet, cela permettrait de fournir à l’étudiant un outil complet pour apprendre la programmation et évaluer son niveau. En plus du projet du chatbot, le système pourrait également s’interfacer avec Webcampus. En effet, intégrer la plateforme courante à Webcampus permettrait de centraliser les services fournis aux étudiants pour les accompagner dans leur apprentissage. Cependant, l’intégration dans les deux cas demanderait beaucoup de temps et reste purement théorique dans le cadre de ce travail.

3.3 Définition des mesures de qualité

La partie la plus importante de ce travail réside dans l’évaluation du programme des étudiants. Pour ce faire, il est nécessaire d’établir une liste de mesures pour chaque critère défini qui seront pris en compte dans notre évaluation.

Cette section décrit les mesures définies pour les critères donnés par la norme ISO/IEC 25000. Pour chaque mesure, sont décrits (1) les mesures de base, (2) la mesure dérivée ainsi que (3) une importance permettant pour chaque critère d’être évaluer pertinemment en ajustant la note selon l’importance de la mesure. L’importance est un chiffre entre 1 et 10, 1 étant le niveau le moins important et 10 le plus important. Le calcul de la note d’un critère est une moyenne pondérée prenant en compte cette importance durant l’évaluation. Aussi, il faut garder

[22] Pattern matching : en informatique, action de vérifier dans une séquence donnée d'éléments la présence des constituants d'un certain pattern.
[23] Expression régulière (regex) : chaîne de caractères qui décrit un ensemble de chaînes de caractères possibles.

en tête le public cible qui sont les élèves en 1ère année de bachelier afin de garder un aspect pédagogique dans la plateforme et ne pas surcharger les élèves d'informations.

3.3.1 Fonctionnalité

La fonctionnalité représente un critère de qualité très important du programme qui permet de déterminer s'il va fonctionner correctement, considérant le code actuel.

Mesure 1 : Exactitude de la syntaxe

Cette mesure consiste à évaluer la syntaxe du code au travers d'une multitude de conventions de base telles qu'un 'continue' dans un 'finally', une mauvaise indentation, du code dupliqué, du code inatteignable... C'est une mesure très importante car la qualité dépend en majeur partie de ces conventions.

Mesures de base		**Exactitude de la syntaxe**
M1		Nombre total de lignes
M2		Nombre d'erreurs syntaxiques basiques détectées
Mesure dérivée		**Ratio d'erreurs syntaxique dans le code** (M2/M1)
Importance		9

Mesure 2 : Exactitude de la sémantique

De la même façon que la mesure précédente, la sémantique du code doit être évaluée. Cependant, il n'est actuellement pas possible, avec les outils utilisés, de faire une telle vérification. Cette mesure n'est donc pas prise en compte dans le cadre de ce mémoire. Elle est néanmoins intéressante à citer car c'est une mesure importante à évaluer et c'est une bonne piste pour les futurs travaux.

Mesures de base		**Exactitude de la sémantique**
M1		Nombre total de lignes
M2		Nombre d'erreurs sémantiques détectées
Mesure dérivée		**Ratio d'erreurs sémantiques dans le code** (M2/M1)
Importance		9

Mesure 3 : Gestion des classes

Une classe ne se gère pas de la même manière qu'une fonction ou autre. Il y a un fonctionnement et des conventions propres aux classes qu'il est nécessaire de respecter. Elles ne sont néanmoins pas présentes dans tous les programmes, c'est pourquoi cette mesure n'est évaluée que si le programme à analyser contient une ou plusieurs classes.

Mesures de base		**Gestion des classes**
M1		Nombre de classes
M2		Nombre d'erreurs relatives aux classes
Mesure dérivée		**Ratio d'erreurs relatives aux classes** (M2/M1)
Importance		7

Mesure 4 : Gestion des exceptions

De même que pour les classes, les exceptions ont leur fonctionnement propre. Elles ne sont pas présentes dans tout programme, la mesure n'est alors évaluée que quand c'est pertinent.

Mesures de base		**Gestion des exceptions**
M1		Nombre de lignes concernant les exceptions
M2		Nombre d'erreurs au niveau de la gestion des exceptions
Mesure dérivée		**Ratio d'erreurs relatives aux exceptions** (M2/M1)
Importance		7

Mesure 5 : Ratio d'erreurs fatales

Un aspect très important de ce projet est sa capacité à corriger les lignes problématiques lors de l'analyse afin que cette dernière puisse s'exécuter sans soucis. Ces lignes problématiques sont les erreurs fatales renvoyées par Pylint sont les erreurs syntaxiques détectées bloquant le processus d'analyse qui sont envoyées à l'algorithme de correction de lignes erronées.

Mesures de base		**Ratio d'erreurs fatales**
M1		Nombre de lignes de code
M2		Nombre d'erreurs fatales rencontrées
Mesure dérivée		**Ratio d'erreurs fatales dans le code** (M2/M1)
Importance		10

3.3.2 Maintenabilité

La maintenabilité est la capacité d'un système à être maintenu, c'est à dire d'être amélioré ou de subir des corrections. Pour ce faire, il est important que le code soit lisible et qu'il respecte certaines conventions pour que quiconque qui le reprend puisse le comprendre et le modifier rapidement.

3.3.2.1 Pertinence des variables utilisées

Les variables sont le cœur du programme, c'est à travers celle-ci que se transmettent les informations au sein du système. Il est important que l'étudiant nomme correctement ses variables avec des noms explicites pour faciliter la prise en main d'un code. De même, des variables définies mais pas utilisées polluent le code et sa compréhension. Et plus problématique, il arrive d'utiliser des variables et d'avoir oublié de les définir ou de les assigner. Ce sont ces erreurs qui sont vérifiées par cette mesure.

Mesures de base		**Pertinence des variables utilisées**
M1		Nombre total de variables
M2		Nombre d'erreurs relatives aux variables
Mesure dérivée		**Ratio d'erreurs relatives aux variables** (M2/M1)
Importance		7

Mesure 1 : Compréhensibilité globale du code

Afin d'assurer la maintenabilité du code, il est nécessaire que celui-ci soit des plus compréhensible. Cela facilite le travail d'implémentation qui sera potentiellement fait plus tard dans le cadre d'une modification par exemple. Cela passe par la réduction de la complexité dans une fonction, une instruction ou le code en général, le fait de ne pas avoir d'éléments inutiles dans le code, etc.

Mesures de base		**Lisibilité du code**
M1		Nombre de lignes de code
M2		Nombre d'erreurs relatives à la lisibilité
Mesure dérivée		**Ratio d'erreurs de lisibilité dans le code** (M2/M1)
Importance		7

Mesure 2 : Respect des conventions d'imports

Il existe des conventions au niveau des imports dans le programme, que ce soit sur l'ordre d'apparition, sur les alias[24], le format ou autre. C'est toutefois un aspect moins important et qui a donc moins de poids dans l'évaluation.

[24] Alias : nom d'emprunt visant généralement à réduire un nom pour faciliter son utilisation.

Mesures de base		Respect des conventions d'imports
M1		Nombre d'imports
M2		Nombre d'erreurs relatives aux imports
Mesure dérivée		**Ratio d'erreurs au niveau des imports** (M2/M1)
Importance		4

Mesure 3 : Respect des conventions de documentation

En Python, il existe des conventions au niveau des docstrings[25]. En effet, celles-ci sont importantes pour faciliter la prise en main d'un code car elles expliquent son comportement. Elles doivent être présentes au niveau du module, des fonctions et des classes. Cependant, il n'est pas pertinent d'évaluer cette mesure sur des petits scripts d'étudiants qui débutent. Elle n'est donc à évaluer que dans le cadre de plus gros projets.

Mesures de base		Respect des conventions de documentation
M1		Nombre de fonctions et classes nécessitant une docstring
M2		Nombre d'erreurs et manquements au niveau des docstrings
Mesure dérivée		**Ratio d'erreurs au niveau des docstrings** (M2/M1)
Importance		5

3.3.3 Sécurité

La sécurité d'un système vise à protéger l'intégrité et la confidentialité des données dans un programme informatique. Évaluer pertinemment la sécurité avec un outil d'analyse statique est complexe. Cependant il est possible de détecter certains éléments dans le code qui sont susceptibles d'amener des problèmes de sécurité. Aussi, les risques ne sont pas évalués de la même façon dans un programme de dix lignes que dans un système plus conséquent. Dans le cadre de ce projet, ce critère est surtout évalué en guise de prévention, pour instaurer une vigilance et responsabiliser les étudiants dans leur développement.

Mesure 1 : Protection contre l'injection de code

Un des fléaux de l'informatique est l'injection de code[26]. En Python, certaines méthodes dites 'built-in', c'est-à-dire qui sont fournies par le langage, peuvent causer des soucis plus ou moins graves si elles sont mal utilisées. Des méthodes telles que 'exec()', 'eval()', 'execfile()' ou 'compile()', qui prennent en argument une chaine de caractère, peuvent être sujettes à des attaques par injection de code si elles prennent une entrée utilisateur non traitée en argument. Il est donc nécessaire de s'assurer que l'étudiant fasse attention et protège ses entrées.

[25] Docstring : littéral de chaîne servant à documenter le code.
[26] Injection de code : exploitation d'une faille pour modifier le comportement d'un programme en insérant des lignes de code malicieuses.

Mesures de base		Utilisation de méthodes problématiques
M1		Nombre de méthodes « builtin » problématiques utilisées
M2		Nombre de méthodes non-sécurisée
Mesure dérivée		**Ratio de failles potentielles** (M2/M1)
Importance		7

3.3.4 Performance

La performance d'un système peut se mesurer aux travers de différents aspects tels que la vitesse d'exécution, l'efficacité, le taux de consommation des ressources, etc., mais ce n'est pas réellement le but de la solution. Néanmoins, il doit être possible de détecter dans le code des lignes qui pourraient causer des baisses de performance, que ce soit par un mauvais agencement du code ou d'une fonction, ou le mauvais placement d'un élément dans le code qui alourdit son exécution. Par exemple, il vaut mieux éviter de mettre des appels de fonction ou des déclarations dans une boucle. Comme ce sont des éléments difficiles à vérifier et qui demanderaient d'utiliser d'autres outils que ceux employés dans ce travail, ce critère n'a pas été implémenté. Des mesures peuvent cependant être proposées dans l'optique d'éventuels travaux futurs.

Mesure 1 : Optimisation des instructions

Comme cité précédemment, cette mesure a pour but d'évaluer la pertinence des éléments du code par rapport à leur performance. Elle n'a pas un grand poids dans l'évaluation car ce n'est pas la plus importante mais il est intéressant de la vérifier.

Mesures de base		Optimisation des instructions
M1		Nombre d'éléments vérifiés
M2		Nombre de problèmes liés à l'optimisation d'instructions
Mesure dérivée		**Ratio d'éléments problématiques** (M2/M1)
Importance		5

3.3.5 Fiabilité

La fiabilité d'un système représente sa capacité à assurer son exécution dans toutes conditions. Dans le cadre de ce mémoire, il n'est pas forcément pertinent d'évaluer ce critère sachant que la plupart des codes évalués sont des petits scripts Python. Cependant, pour des programmes plus conséquents il pourrait être intéressant de vérifier leur fiabilité.

Mesure 1 : Robustesse du programme

Dans le cas où le programme attend des arguments, ou qu'il intègre des entrées d'utilisateurs, il pourrait être intéressant de tester sa robustesse en insérant des valeurs arbitraires et en comparant les résultats pour voir si le programme gère tous les cas de figure. Ces valeurs

pourraient provenir d'une liste dans laquelle se trouvent différents types de valeurs possibles à entrer, testant pour chaque type de donnée différentes limites du domaine.

Mesures de base		**Robustesse du programme**
M1		Nombre d'entrées testées
M2		Nombre d'entrées robustes
Mesure dérivée		**Ratio d'entrées robustes** (M2/M1)
Importance		4

3.3.6 Utilisabilité

L'utilisabilité représente la capacité du système à être compris et utilisé de manière efficace par les utilisateurs. À travers ce critère, il convient d'évaluer les interactions de différents utilisateurs avec le programme afin de voir si celui-ci est compréhensible, facile à prendre en main, attractif... Mais un tel critère nécessite forcément une interaction avec des utilisateurs ainsi que des analyses de ces interactions pour pouvoir être évalué ; il est donc difficile d'automatiser ce processus. De plus, la plupart des programmes analysés dans le cadre de ce projet sont des scripts qui ne demandent pas de telles interactions. Il n'est donc pas pertinent d'évaluer ce critère connaissant le public cible, ce n'est pas le but visé.

3.3.7 Portabilité

La portabilité d'un système est sa capacité à être transféré d'un environnement à un autre. Sur base de ce changement de milieu, plusieurs éléments peuvent être observés comme son adaptabilité au nouvel environnement, sa coexistence avec les éléments déjà présents dans le système... À nouveau, il n'est pas pertinent d'évaluer ce critère connaissant le public cible et le type des programmes évalués.

3.4 Implémentation

À travers ce point est décrite l'implémentation de la solution, c'est à dire les choix techniques et les éléments mis en place pour assurer le bon fonctionnement de la plateforme.

3.4.1 Maquettes

L'Annexe 1 décrit les maquettes qui ont été faites avant de se lancer dans l'implémentation, afin de définir les visuels souhaités. Il est toujours important de faire des maquettes étant donné que cela est plus rapide à faire et à modifier que du code. Cela permet d'avoir une ligne directrice et offre un gain de temps considérable pour l'implémentation de la plateforme, à condition de s'y référer pendant le développement.

3.4.2 Base de données

Un schéma de base de données a également été mis en place de façon à couvrir tous les besoins du système à développer. Le schéma conceptuel de la base de données est présenté à la Figure 20, il reprend les différents composant nécessaires au bon fonctionnement du système.

Parmi ces éléments se trouvent (1) les éléments propres à l'aspect de gamification, symbolisés par les tables 'Ranking', 'Reward '..., (2) les éléments propres à la gestion de l'utilisateur (la table 'Student' qui hérite de 'Django_User'), et (3) les éléments propres à l'évaluation du programme, la table 'Exercice' étant l'élément central avec d'autres tables comme 'Criterion', 'Measure' et 'Programm_error' par exemple.

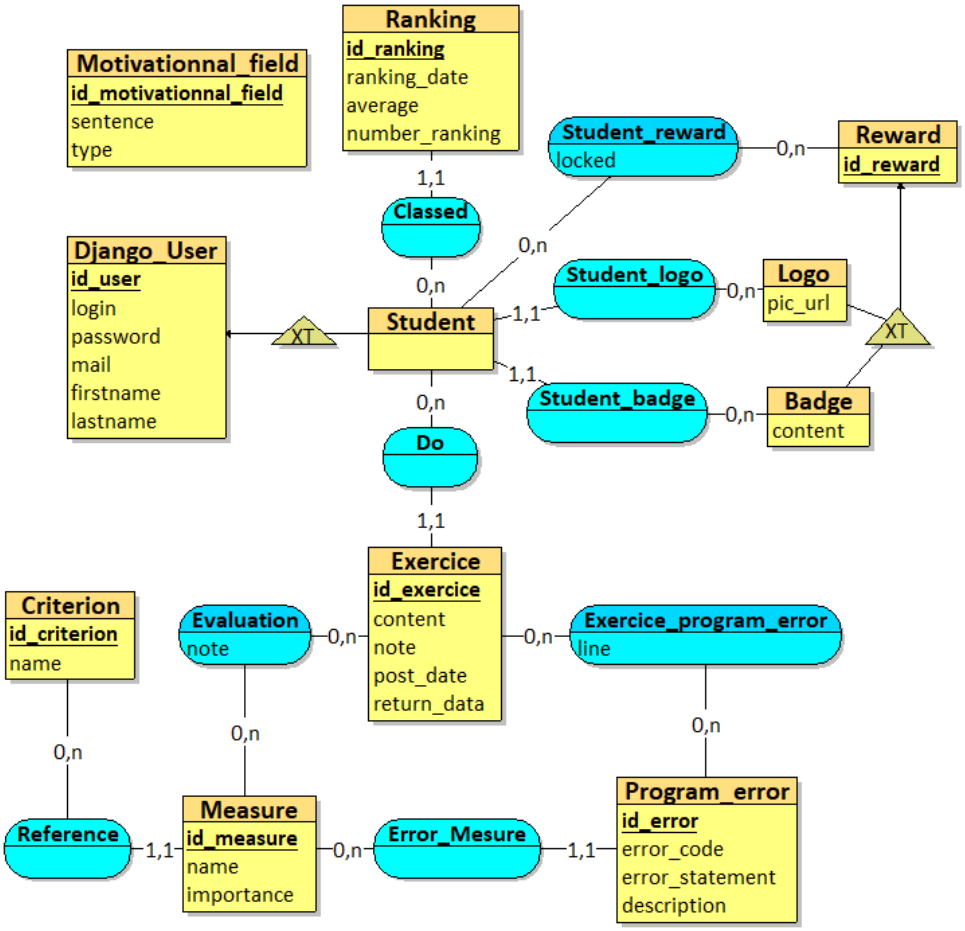


Figure 20 : schéma entité-association (E.-A.) de la base de données du système

La Figure 21 montre le schéma relationnel qui découle du schéma E.-A, et qui modélise plus concrètement les relations entre les données. L'Annexe 2 propose une description des tables présentes afin d'aider à comprendre le système au mieux.

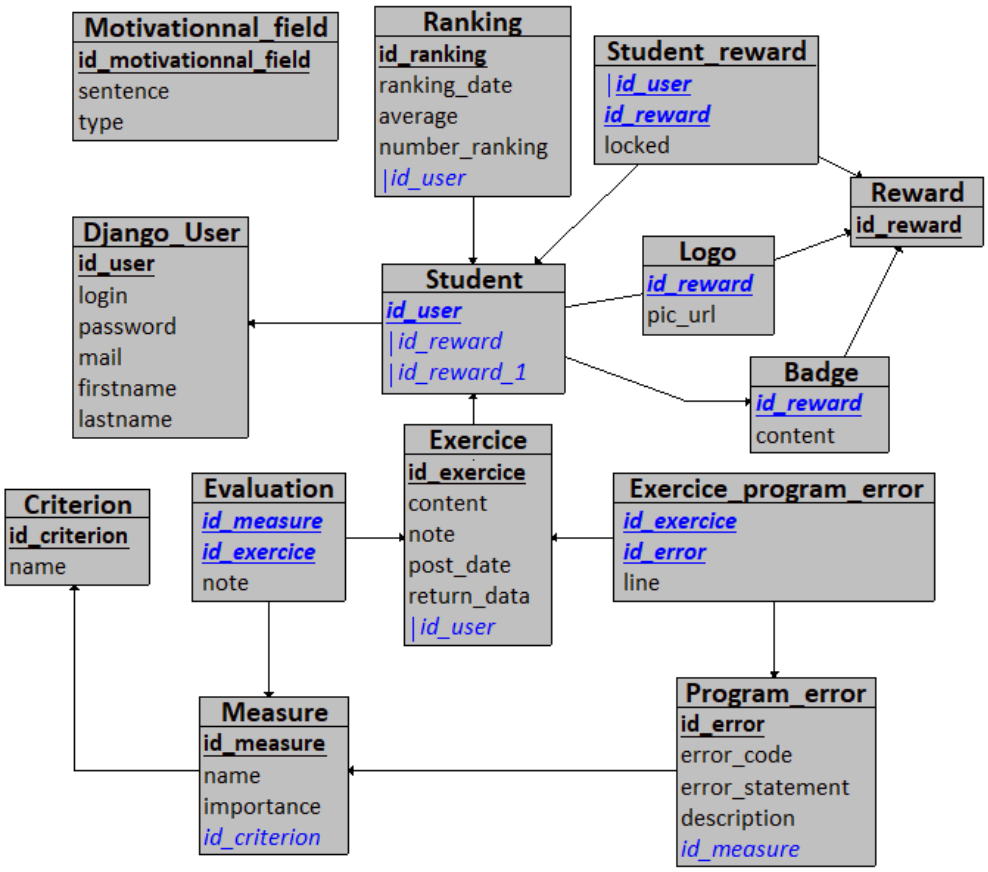


Figure 21 : Schéma relationnel de la base de données du système

3.4.3 Gamification de la plateforme

Le système proposé offre plusieurs aspects de gamification. Tout d'abord, la Figure 22 montre différentes statistiques que la plateforme permet aux utilisateurs de voir. Sur son profil, l'étudiant peut voir son score moyen et son évolution dans le temps. Il est également possible de voir les différentes moyennes pour chacun des critères qui composent un exercice. Tous ces éléments permettent à l'élève de se situer dans le processus d'apprentissage, de voir où il a des lacunes, où il s'est amélioré... et donc de satisfaire un éventuel besoin de reconnaissance puisque les autres étudiants peuvent également voir ces statistiques.

Figure 22 : Statistiques d'un utilisateur dans la plateforme METAssistant

Deuxièmement, comme montré sur la Figure 23, la plateforme introduit la notion de compétitivité par le biais d'un classement entre les étudiants. Ce classement présente les élèves dans l'ordre décroissant par rapport à leur moyenne générale. Il donne aux élèves un but à atteindre et permet aux élèves de se situer par rapport aux autres et donc de les motiver à pratiquer plus d'exercices afin de grimper dans le classement. Même si l'aspect de compétitivité peut booster les étudiants à devenir meilleur, il peut également avoir un effet néfaste d'une part pour les personnes à comportements addictifs et va potentiellement ajouter de la toxicité dans certaines relations entre élèves d'autre part. C'est donc un aspect intéressant à implémenter, mais il vaut mieux éviter de trop le développer ou qu'il ne soit trop présent afin de s'assurer que la plateforme reste bienveillante et ainsi éviter les dérives.

Figure 23 : Système de classement dans la plateforme METAssistant

Le dernier aspect de la gamification est la notion de récompenses. La Figure 24 concerne la partie récompenses du profil utilisateur. En fonction de sa progression, du nombre d'exercices réalisés, de sa moyenne, etc., l'élève se verra attribuer certaines récompenses comme des badges ou des logos pour personnaliser son profil, qui seront visibles par les autres utilisateurs dans le classement ou la page de profil. Ce système introduit un certain besoin d'achèvement, car l'étudiant voudra collecter toutes les récompenses possibles sur la plateforme. Cet aspect est renforcé par l'utilisation de barres de chargement qui permettent aux élèves de se situer dans leur progression globale.

Figure 24 : Récompenses et barres de progression dans la plateforme METAssistant

3.5 Résumé

Ce point décrit la mise en place de la solution par rapport à la section 2. Afin de la mettre en place, il est nécessaire de définir un cadre de travail qui va guider l'implémentation. L'objectif est donc une plateforme permettant une évaluation formative tout au long de l'année, avec une méthode d'évaluation du code orientée qualité, et ce dans un environnement ludique propice à booster la motivation des élèves. Pour ce faire, des outils et technologies sont déterminés de façon à répondre au mieux aux exigences. Les langages de programmations Python et Javascript sont ainsi les technologies les plus pertinentes à utiliser (par l'intermédiaire respectivement des framework Django et Vue.js). Aussi, afin de pouvoir utiliser un outil d'analyse statique pour tout code, même syntaxiquement incorrects, un algorithme de correction de lignes est proposé et permet de modifier les lignes erronées de manière à rendre le code lisible par l'analyseur.

Les mesures définies pour les différents critères de la norme ISO/IEC 25000 permettent de structurer l'évaluation et de la répartir entre différents aspects de la programmation afin d'avoir une vue d'ensemble sur la qualité du code analysé et fournir une note et un feedback pertinent à l'étudiant. Cependant, tous les critères de cette norme ne sont pas pertinents à évaluer dans le cadre de ce projet, et toutes les mesures définies n'ont pas pu être implémentées actuellement. L'implémentation du système est décrite notamment par les maquettes et les différents schémas représentant les données. Pour finir, les différents aspects de gamification implémentés dans la plateforme permettent d'ajouter beaucoup à l'attrait de la plateforme, via des récompenses, un classement… afin de rendre les utilisateurs plus investis dans leur apprentissage.

4 Validation de la solution

Après avoir mis en place cette solution, il est nécessaire de tester celle-ci afin de s'assurer de son bon fonctionnement et tester sa pertinence en tant qu'évaluateur.

La crise sanitaire ayant rendu les interactions avec les professeurs et élèves sont plus compliquées qu'en temps normal, il est donc difficile de proposer une validation exhaustive de la solution. De plus, la validation d'un tel projet est plus pertinente si elle est lancée en version de test pour un groupe d'étudiants pendant une durée donnée afin de comparer les performances avec les années précédentes et d'autres élèves.

Afin de proposer une validation réalisable malgré ces contraintes, des exemples de programme réalisés par les étudiants dans le cadre du cours « *INFOB131 Introduction à la programmation* » ont été anonymisés et mis à disposition, ce qui a permis de bien comprendre la problématique dans un premier temps, et de tester le système dans un second. Dans cette section, les évaluations de deux codes, un court et un plus long, sont décrites pour illustrer le fonctionnement de la plateforme et expliquer les détails de l'évaluation. Les résultats des tests sont ensuite analysés.

4.1 Méthodologie de test

Cette section reprend deux exemples de programmes, un court et un plus long afin de montrer comment la solution réagit et de décrire le retour qu'il donne. Ces deux types de programmes sont différenciés car les petits codes de quelques lignes servent généralement à mettre l'accent sur des aspects spécifiques de la programmation. Il n'est pas pertinent d'évaluer la totalité des aspects pour ce type de programme, par exemple la portabilité et l'utilisabilité ne sont pas forcément pertinents à évaluer pour un petit script. En revanche il est intéressant de vérifier certaines bonnes pratiques et s'assurer de sa fonctionnalité. Les programmes plus longs sont généralement pour des projets plus conséquents, de ce fait il est important de vérifier qu'un maximum d'aspects soient vérifiés afin d'être le plus complet possible dans le feedback.

À noter que toutes les mesures définies dans la section 3.3 ne sont pas implémentées. Avec la contrainte de temps, seuls certains aspects des critères de fonctionnalité, maintenabilité et de sécurité ont été implémentés. À l'avenir, il faudra soit ajouter d'autres mesures, si cela paraît pertinent, pour compléter l'évaluation, soit raffiner les mesures existantes et les améliorer pour qu'elles soient plus pertinentes. Cependant, les mesures définies dans ce papier forment une bonne base sur laquelle s'appuyer pour commencer.

4.1.1 Évaluation d'un programme court

Les programmes courts sont généralement donnés aux étudiants en début d'apprentissage afin de les aider à se familiariser avec la syntaxe d'un langage (dans ce cas-ci Python). C'est donc dans ce type d'exercice qu'il y a le plus de risques de retrouver des erreurs qui rendront l'évaluation du programme difficile car le code ne pourra souvent pas être exécuté. L'ACL mis en place dans ce travail permet de corriger ce type d'erreurs et donc de rendre le code exécutable afin d'y appliquer l'évaluation de la qualité.

```
1  import os
2  import sys
3  import random as random
4
5  df write_random_pow(number):
6      r = random.randint(0, 10)
7      random_pow = pow(number, r)
8
9      "Random pow: " = s
10
11     with os.open('random_pow.txt', None, 'w') as file:
12         file.write(s + random_pow)
13
14     return random_pow
```

Figure 25 : Exemple de code d'étudiant soumis pour évaluation

La Figure 25 présente un exemple de code court donné au système pour évaluation. Des erreurs ont été délibérément insérées pour l'exemple. Au début, l'exécution de Pylint entraînera une erreur. En fait, les erreurs aux lignes 5 et 9 sont des erreurs syntaxiques que tout analyseur statique ne comprendra pas, ce sont les erreurs dites fatales. Chacune sera envoyée à l'algorithme de correction de lignes qui suggérera une correction en tenant compte de l'intention détectée derrière la ligne. À la ligne 5, il détectera l'intention de définir une fonction, et remplacera la faute de frappe "df" par "def". À la ligne 9, il détectera l'intention de faire une assignation et réordonnera la ligne avec la variable au début et sa valeur à la fin. Ces deux lignes étant ajustées, Pylint peut maintenant commencer à analyser le contenu et retourner les codes d'erreur liés au programme donné. Les erreurs aux lignes 2 et 3 pour l'importation inutilisée et le même nom d'alias, ou aux lignes 6 et 9 pour les noms de variables invalides, ainsi que les deux erreurs fatales rencontrées, sont considérées pour l'évaluation.

Tableau 4 : Détail de la cotation pour le code en Figure 25

Critère	Mesure	Note (/20)
Fonctionnalité	Exactitude de la syntaxe	20
	Ratio d'erreurs fatales	16.6
Maintenabilité	Pertinence des variables	16.9
	Respect des conventions d'imports	13.3
	Lisibilité du code	19.13
Total :		17.6

Le Tableau 4 résume les détails de l'évaluation pour chaque mesure. Notez que, pour un programme, seules les mesures concernées sont prises en compte pour le score final, qui est la moyenne des mesures ci-dessus. De cette façon, même s'il y a des erreurs à certains endroits, la prise en compte de plusieurs aspects de la programmation est moins punitive et récompense l'étudiant pour ses efforts. Il est important pour les débutants d'avoir les détails de chaque mesure afin qu'ils sachent quelles erreurs ils ont commis et ce qu'ils ont fait de bon, cela les motive davantage. Il est également intéressant de noter qu'il y a une erreur à la ligne 12, essayant

de concaténer une chaîne de caractères avec un nombre entier, qui ne sera pas détectée en raison de l'utilisation d'un analyseur de code statique car il s'agit d'une erreur sémantique.

4.1.2 Évaluation d'un programme plus long

Les programmes de plus grandes tailles sont généralement des exercices plus poussés comme des projets soumis par les étudiants. Ce type d'exercice est plus susceptible d'arriver en fin de quadrimestre et il est donc nécessaire de vérifier que tous les principes de programmation sont respectés.

De manière générale, les projets sont moins enclins à contenir des erreurs syntaxiques graves bloquant le processus. Notamment parce que les étudiants ont plus d'expérience qu'au début de l'année, mais ils passent également plus temps sur le programme et doivent s'assurer qu'il s'exécute bien.

```python
import os
import sys

class SampleClass:
    """
    """
    def __init__(id, description):
        self.id = id
        self.description = description

def input_pow(number):
    """
    calculate the number's power value and write it to a file

    Parameters:
    ----------
    number: the number to calculate the power

    Return:
    ------
    input_pow : the power of the given number
    """
    p = input("Type the power :")
    input_pow = pow(number, eval(p))

    with os.open('input_pow.txt', None, 'w') as file:
        file.write(input_pow)

    return input_pow
```

Figure 26 Code plus conséquent soumis à l'algorithme d'évaluation

La Figure 26 montre un code plus long soumis pour évaluation. Ce code est délibérément plus concis qu'un projet complet, il est uniquement utilisé à titre d'exemple. Ici, Pylint peut s'exécuter sans rencontrer de problèmes. En effet, les programmes plus longs sont souvent des

projets donnés au second quadrimestre, dans lesquels se trouvent généralement moins d'erreurs syntaxiques graves et bloquantes pour le processus.

Pour cet exemple, Pylint renvoie différentes erreurs au niveau des imports, de la classe, des docstrings, des variables ainsi que de la sécurité. Tout d'abord, l'import à la ligne 2 n'est pas utilisé dans le programme, ce qui n'est pas une erreur très grave, mais qui peut nuire à la compréhension et donc la maintenabilité d'un code à long terme (et est donc puni à moindre mesure au niveau de la lisibilité du code). Ensuite, par rapport à la docstring vide à la ligne 5, il est toujours nécessaire de garder en tête qu'il est important de bien documenter son code ; c'est donc puni au niveau des conventions de documentation. À la ligne 7, l'argument 'self' est manquant, mais est en plus utilisé par la suite. Cette erreur est comptée au niveau de la gestion des classes, comme c'est une erreur propre aux classes. La variable 'p' déclarée à la ligne 23 ne respecte pas les conventions d'appellation et réduit la note au niveau de la pertinence des variables. Deux problèmes surviennent en ligne 24, avec premièrement une redéfinition d'un nom extérieur causée par la variable 'input_pow' du même nom que la fonction dans laquelle elle se trouve. Et secondement la méthode 'eval()' utilisée sur un input d'utilisateur, qui consiste en une faille de sécurité assez importante.

Tableau 5 : Détail de la cotation pour le code en Figure 26

Critère	Mesure	Note (/20)
Fonctionnalité	Exactitude de la syntaxe	19.13
	Gestion des classes	18
Maintenabilité	Pertinence des variables	17
	Respect des conventions de documentation	13.33
	Lisibilité du code	19.13
Sécurité	Utilisation de méthodes problématiques	15
Total :		17.21

Le Tableau 5 résume les détails de l'évaluation du code précédemment décrit, pour chaque mesure relative. Beaucoup d'erreurs ont été détectées, mais qui sont moins importantes pour la plupart. Encore une fois, le système récompense l'étudiant pour ses efforts et tend à ne pas trop punir les petites fautes.

4.2 Analyse des résultats

Il est difficile de faire des déductions sur base de ces exemples, tant l'échantillon est petit. Néanmoins, il est possible de faire quelques observations et de voir les différences entre des petits codes et des plus imposants. Tout d'abord, l'exemple court permet de montrer l'absence de notion de sémantique dans l'évaluation. En effet, cet aspect est très difficile à évaluer et les technologies utilisées durant ce projet ne permettent pas de réaliser une telle tâche. En revanche, cet exemple montre que malgré la présence d'erreurs syntaxiques bloquant l'analyseur statique au premier abord, il est possible de passer outre ce problème en corrigeant les lignes problématiques à l'aide de l'ACL, permettant ainsi de procéder à l'évaluation du code. Naturellement, les erreurs corrigées par l'ACL sont comptabilisées et prises en compte dans la note finale. Ensuite, les résultats montrent également que les mesures évaluées sont adaptées

au programme analysé. Par exemple, la gestion des classes n'est pas évaluée dans le premier exemple car le programme n'en contient pas. Aussi, les notes globales des deux exemples sont assez similaires malgré un plus grand nombre d'erreurs dans le grand code. En effet, la note finale dépend du code et de ses attributs ; il y a beau avoir plus d'erreurs dans le second exemple, le code est également plus grand et il y a plus d'éléments à évaluer que dans le premier, il est donc normal que la note soit proportionnelle à cela. De plus, l'importance des erreurs détectées permet de mettre l'accent sur les erreurs plus critiques et d'ajuster la note en fonction de ça.

4.3 Résumé

À travers ce point, deux types de programmes ont été soumis en test au système, un court et un plus poussé. Cela permet de voir comment la solution réagit et s'adapte en fonction du code analysé. L'aspect d'adaptabilité est notamment poussé par l'algorithme de correction de lignes ainsi que l'application de certaines mesures en fonction des attributs du code. Dès lors, le retour fourni à l'élève et la note qui en découle sont cohérents. Malheureusement, la validation ne tiens pas compte d'assez d'éléments pour être vraiment pertinente. Il aurait été intéressant d'évaluer un plus grand échantillon de programmes différents pour analyser les retours de la plateforme et potentiellement les comparer avec des vraies corrections de professeurs pour situer la qualité de l'évaluation de la solution.

5 Travaux futurs

Dans ce travail sont décrit les principales méthodes pour évaluer le code d'un étudiant. Ensuite est expliqué la solution ‘METAssistant’ et son application sur un petit échantillon pour l'illustrer. L'objectif est de fournir le feedback le plus pertinent possible pour aider les élèves à approfondir leurs connaissances. Ce feedback prend en compte différentes mesures, liées aux critères de qualité fournis par la norme ISO/IEC 25000, définis pour évaluer le plus précisément possible les programmes des étudiants, en tenant compte de tous les aspects de la programmation. La solution fournit également un retour complet à l'étudiant et vise à stimuler l'intérêt et l'engagement de ses utilisateurs par le biais d'une plateforme ludique. Elle offre une base pour évaluer les codes d'étudiants voir même de personnes plus confirmées, mais il peut être amélioré à différents niveaux.

5.1 Affinage et complétion des mesures de qualité définies

Plutôt dans ce travail, ont été définis des critères de qualité ainsi que les mesures nécessaires pour calculer une note sur le critère en question. Par contrainte de temps, l’accent a été mis sur certaines mesures et critères au dépend d’autres. Les mesures définies doivent être affinées et complétées. Le système met actuellement en œuvre un ensemble de mesures portant principalement sur les critères de fonctionnalité, de maintenabilité et de sécurité, pour la plupart grâce à Pylint. Mais il lui manque l'analyse sémantique et de nombreuses autres mesures. Celles-ci doivent être améliorées et complétées par des mesures tenant compte de la norme ISO/IEC 25000 et éventuellement par l'utilisation d'autres outils pour approfondir l'analyse. Cela permettrait de fournir un retour plus poussé sur la qualité du code en évoquant des aspects comme la sécurité, l’utilisabilité...

5.2 Une validation plus poussée

L’étape de validation actuelle se base sur deux exemples de codes et la comparaison de leurs résultats. Le problème est qu’il faudrait faire des tests sur un plus grand échantillon de programmes pour avoir plus de données, l’analyse serait alors plus pertinente. Il pourrait également être intéressant de comparer les résultats obtenus par la plateforme avec des résultats d’évaluations de professeurs pour pouvoir situer la pertinence de la solution. Aussi, il serait bon de tester la plateforme en condition réelle pendant un quadrimestre ou une année, pendant laquelle les utilisateurs pourraient faire des retours de manière continue. Sur base de ce test, les utilisateurs pourraient également remplir un questionnaire et certains pourraient être interviewés afin d’avoir un retour complet sur la plateforme, ses qualités et les points à améliorer selon les principaux concernés.

5.3 Amélioration de l’aspect ludique

L'aspect ludique peut être poussé plus loin. D'autres fonctions de gamification peuvent contribuer à accroître l'implication et la capacité d'apprentissage des étudiants. Il pourrait donc être intéressant d’intégrer de la gamification au sein de l’élément central du système à savoir l’évaluation du programme. Comme vu au point 2.3.3.1, Sleuth propose une interface qui immerge l’utilisateur dans une enquête policière et qui sert de façade pour apprendre la

programmation à des élèves. Mettre en place un système semblable au sein du METAssistant aurait pour but d'impliquer plus l'étudiant afin qu'il s'exerce encore plus.

Une attention pourrait également être portée sur une plus grande personnalisation du système. Par exemple, modifier la disposition de l'interface, ses couleurs, ses dimensions... pour gagner en attractivité. La Figure 27, reprise de Moodle montre une telle interface permettant de disposer les éléments à sa guise.

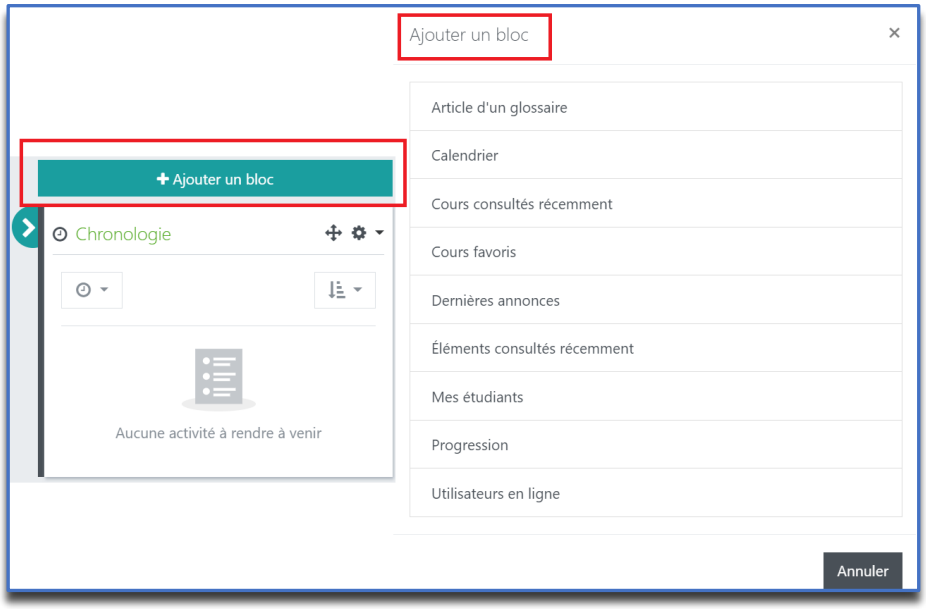


Figure 27 : Exemple de personnalisation de l'interface Moodle[27]

5.4 Visualisation de l'information

La plateforme METAssistant permet aux étudiants et professeurs de consulter des données en tout genre comme la moyenne générale, la moyenne dans certains critères, etc. Actuellement ces informations sont affichées sous forme de texte ce qui dans le cadre de la visualisation de données reste très basique. Se pencher sur des aspects avancés de la visualisation de données pourrait être intéressant afin de fournir aux parties prenantes des informations plus visuelles et ainsi aider à mieux comprendre les informations. Par exemple, le 'radar chart' est un type de graphique qui permet de quantifier un ensemble de données sur un plan en 2 dimensions[28]. Il pourrait être pertinent afin d'améliorer la visualisation de la moyenne par critère (fonctionnalité, maintenabilité...).

[27] Personnalisation de Moodle depuis l'interface Webcampus, repris de https://webcampus.unamur.be/my/index.php.
[28] Radar chart : https://www.data-to-viz.com/caveat/spider.html.

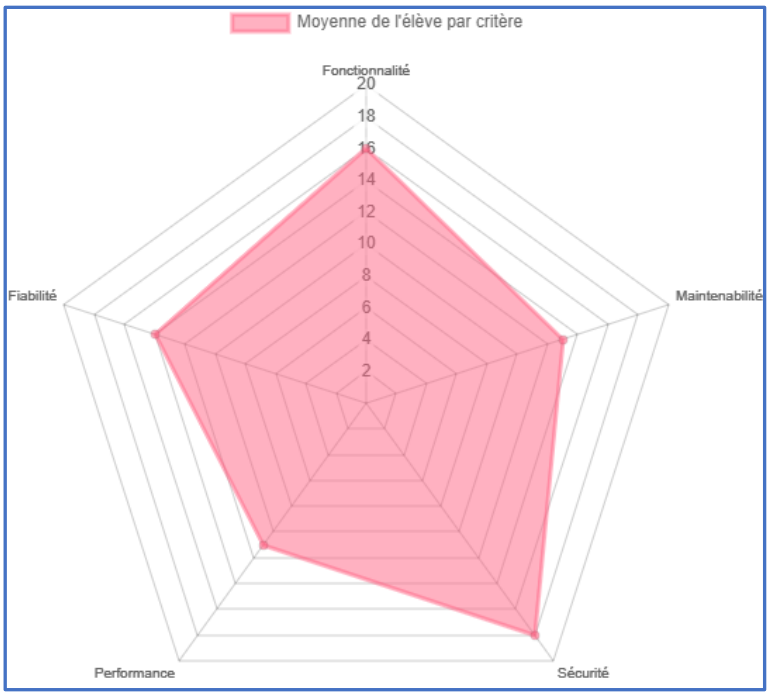


Figure 28 Exemple de radar chart de la moyenne d'un élève pour chaque critère[29]

La Figure 28 illustre un radar chart avec 5 critères de qualité différents. Il permet de visualiser directement pour quels critères l'étudiant est bon ou moins bon, et cela ajoute une touche visuelle au profil. L'intégration de ce genre d'élément de visualisation au sein de la plateforme apporterait ainsi une meilleure prise de connaissance des informations de la part des étudiants et leurs permettraient ainsi de mieux se situer dans leur apprentissage.

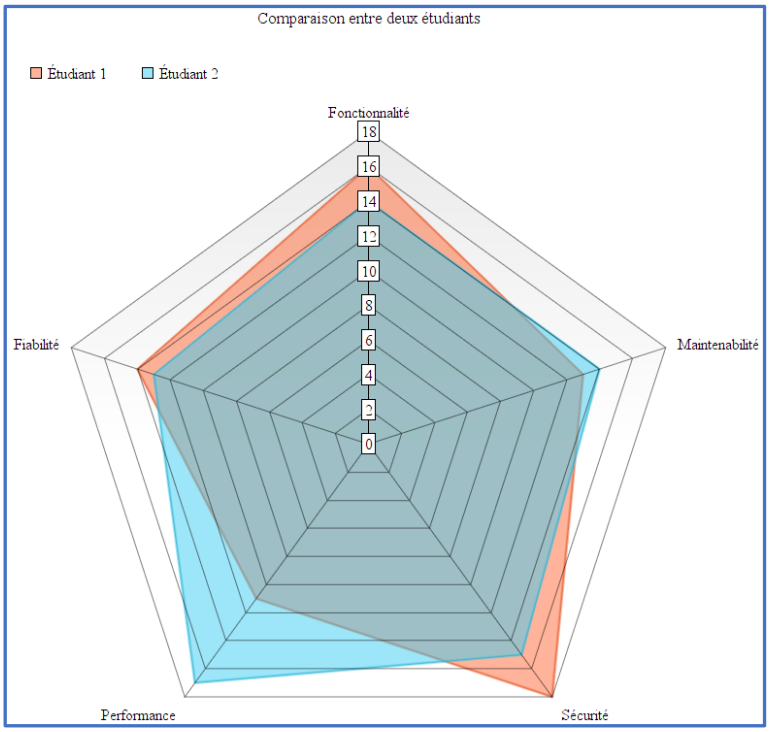


Figure 29 : Radar chart comparant les notes de deux élèves par rapport à des critères de qualité[30]

[29] Produit depuis https://chachart.net/radar?lang=en.
[30] Produit depuis https://www.onlinecharttool.com/graph?selected_graph=radar.

La Figure 29 montre un radar chart avec les mêmes critères de qualité. Ce type de graphique permettrait donc également de comparer deux ou plusieurs élèves entre eux. À nouveau, cela peut être intéressant, bien qu'à prendre avec des pincettes sachant que l'aspect de compétitivité ne doit pas être le focus principal de la plateforme afin d'éviter les dérives. Naturellement, il y a d'autres moyens et types de graphiques pour améliorer la visualisation de l'information au sein de la plateforme. C'est un aspect intéressant à développer sachant qu'il permet de comprendre et assimiler plus facilement des informations et augmente l'aspect ludique de la plateforme.

5.5 Amélioration de l'algorithme de correction de lignes

L'algorithme de correction de lignes proposé pour les erreurs fatales peut être assimilé à un interpréteur avec la mise en correspondance d'une expression régulière créée à partir de la ligne d'erreur et de l'expression régulière de la syntaxe. Même si la correspondance est améliorée par certains outils comme la distance de Damerau-Levenshtein et l'algorithme de correspondance, elle manque toujours de précision pour trouver le bon numéro de ligne d'erreur ou pour corriger les erreurs sur plusieurs lignes. Une solution pourrait être d'utiliser une architecture de "long short-term memory" (LSTM) ou de "recurrent neural network" (RNN). Une recherche a déjà été menée sur ce sujet pour le langage Java (**Hindle, Patel, Santos, Amaral et Campbell, 2018**). Cette recherche exploite les modèles de langage 'n-gram' et LSTM afin de localiser les erreurs syntaxiques dans un code et proposer une solution. La problématique de base étant que pour les erreurs syntaxiques telles qu'une accolade manquante en fin d'un 'if', le compilateur renvoie des erreurs par rapport aux lignes qui suivent la ligne en question. L'erreur est donc mal localisée et il est compliqué de la corriger. Les résultats montrent que seul un cas sur deux est bien corrigé et qu'il faudrait approfondir la recherche et la complémenter avec d'autres méthodes, mais que c'est néanmoins possible et que cela permet d'effectivement trouver et corriger des erreurs syntaxiques qui ne sont pas bien localisées par le compilateur.

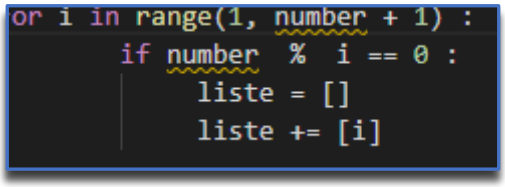


Figure 30 : Exemple d'erreur sémantique courante

La Figure 30 montre une erreur sémantique courante qu'un analyseur classique ne peut pas ou difficilement détecter. Il serait intéressant de pousser la recherche plus loin pour voir si une IA pourrait détecter ce genre d'erreur, mais aussi la corriger…

5.6 Résumé

Diverses pistes d'améliorations ont été proposées pour le projet METAssistant. En effet, en considérant les contraintes de temps et de ressources disponibles durant le stage, des priorités ont dû être fixées, que ce soit pour la définition des mesures, la validation de la plateforme ou l'implémentation de certaines fonctionnalités. Les pistes évoquées ici permettent de mettre l'accent sur des éléments importants et à prioriser dans le cadre de travaux futurs, que ce soit au niveau du frontend avec la visualisation des données ou au niveau du backend avec l'amélioration de l'algorithme de correction ou le raffinage des mesures.

6 Conclusion

L'apprentissage de la programmation demande un travail rigoureux. Les étudiants doivent se confronter à une nouvelle façon de penser, de visualiser un problème afin de le résoudre de manière adéquate, ce qui n'est pas évident et entraine souvent des abandons. La solution décrite dans ce mémoire propose une plateforme dans laquelle un étudiant peut faire évaluer son code d'un côté, et où un professeur peut avoir un aperçu du niveau de compréhension de la matière par ses élèves de l'autre. Pour ce faire, diverses recherches ont été effectuées afin de trouver des pistes de solution pour une évaluation pertinente. Elles ont été menées sur base des questions de recherche définies pour ce projet.

Premièrement, en ressortent divers facteurs à prendre en compte pour l'évaluation d'un programme. À l'aide de la norme ISO/IEC 25000, des mesures ont été définies pour les différents critères qu'elle propose. Elles ont pour but de couvrir un maximum les aspects de la programmation afin d'avoir une évaluation précise et complète, orientée sur la qualité du code. Aussi, la plateforme s'inscrit dans un type d'évaluation formative, permettant un suivi des étudiants tout au long du processus d'apprentissage et un approfondissement de leurs connaissances grâce à un retour clair et précis.

Secondement, afin d'évaluer ces mesures définies plus tôt, un outil d'analyse statique est utilisé. Sachant que les codes analysés proviennent majoritairement de débutants en programmation, il est fréquent d'y retrouver des erreurs syntaxiques bloquant le processus d'analyse de cet outil. Pour pallier ce manque de flexibilité, un algorithme de correction de lignes est proposé. Il est basé sur le 'pattern matching' et considère une marge d'erreur afin d'ajuster les lignes erronées, permettant ainsi à l'analyse de s'exécuter correctement.

Dernièrement, comme dit précédemment, la programmation est un domaine complexe. Un retour complet et précis sur la qualité d'un code est une bonne chose pour aider les élèves, mais ce n'est pas suffisant pour maintenir leur intérêt à s'exercer. Pour les inciter à continuer de s'entrainer et d'améliorer leurs compétences, des aspects de gamification sont implémentés dans la plateforme. Ils permettent notamment de donner un objectif supplémentaire aux étudiant par le biais de récompenses et de cet aspect de progression qu'elles insufflent. Aussi, un classement et l'accès à des statistiques pour les élèves leur permet de se confronter entre eux et leur donne une raison supplémentaire pour s'améliorer et devenir meilleur que les autres.

L'étape de validation a permis de voir une certaine adaptabilité au niveau de l'évaluation d'un code, avec un retour cohérent sur les notes attribuées. Cependant, l'échantillon de programmes testés n'est pas suffisamment grand pour pouvoir juger de la pertinence de la solution. C'est pourquoi une validation plus poussée est évoquée dans les travaux futurs.

À travers ces différents éléments, METAssistant tente d'apporter une évaluation pertinente, dans une plateforme accueillante favorisant un entrainement continu des élèves et augmentant leur implication dans leur apprentissage de la programmation. De plus, la plateforme possède un grand potentiel d'amélioration. Tout cela fait de cette solution une base intéressante à exploiter dans des travaux futurs, qui pourrait à termes devenir un outil indispensable dans les études en informatique.

7 Références

[1] Abran A., Al-Qutaish R., Desharnais J.-M. & Habra N., “An Information Model for Software Quality Measurement with ISO Standards”, Proceedings of the International Conference on Software Development (SWDC'05), pp.104-116, mai 2005.

[2] Berlioz A., “Approche systémique à l'école. L'itinéraire d'une rééducatrice de la « réparation » à la mobilisation du pouvoir parental”, *Thérapie Familiale*, vol.28, no.2, pp.153-166, janvier 2007.

[3] Christanti V.M., Rudy & Naga D.S., “Fast and Accurate Spelling Correction Using Trie and Damerau-levenshtein Distance Bigram”, *TELKOMNIKA*, vol.16, no.2, pp.827-833, avril 2018.

[4] Curonici C., & McCulloch P., “L'approche systémique en milieu scolaire : réflexions 20 ans après”, *Thérapie Familiale*, vol.25, no.3, pp.381-405, janvier 2004.

[5] Derval G., Gego A., Reinbold P., Frantzen B. & Van Roy P., “Automatic grading of programming exercises in a MOOC using the INGInious platform”, *European MOOC Stakeholder Summit 2015 (EMOOCs 2015)*, pp.86-91, 2015.

[6] Deterding S., Dixon D., Khaled R. & Nacke L., “Du game design au gamefulness : définir la gamification”, *Sciences du jeu*, vol.2, no.2, mai 2015.

[7] Earle C., Fredlund L.-Å. & Hughes J., “Automatic Grading of Programming Exercises using Property-Based Testing”, *The 2016 ACM Conference*, pp.47-52, juillet 2016.

[8] Fenton N. & Bieman J., *Software metrics: a rigorous and practical approach*, CRC press, Floride, États-unis, octobre 2014, 617p.

[9] Karsenti T. & Bugmann J., “ASPID : un modèle systémique des usages du numérique en éducation”, *Le numérique*, pp.47-62, septembre 2018.

[10] Katan S. & Anstead E., “Work In Progress: Sleuth, a programming environment for testing gamification”, *IEEE Global Engineering Education Conference (EDUCON)*, avril 2020.

[11] Knutas A., Ikonen J., Uolevi N. & Porras, J., “Increasing collaborative communications in a programming course with gamification: A case study”. *ACM International Conference Proceeding Series*, vol.883, juin 2014.

[12] Labouidya O., Elkamoun N. & Khouilid M., “Les concepts de l'approche systémique appliqués à l'évaluation de l'enseignemen”, *26ème colloque de l’ADMEE Europe*, pp.1-10, janvier 2014.

[13] Laveault D., “De la ‘régulation’ au ‘réglage’: élaboration d'un modèle d'autoévaluation des apprentissages”, *Régulation des apprentissages en situation scolaire et en formation*, pp.207-234, 2007.

[14] Le Robert, *Le Petit Robert de la Langue Française 2020*, Dictionnaires Le Robert, Paris, France, mai 2019, 2880 p.

[15] Lee J. & Hammer J., “Gamification in Education: What, How, Why Bother?”, *Academic Exchange Quarterly*, vol.15, no.2, pp.1-5, janvier 2011.

[16] Martin R. & Shafer L., “PROVIDING A FRAMEWORK FOR EFFECTIVE SOFTWARE QUALITY MEASUREMENT: MAKING A SCIENCE OF RISK ASSESSMENT”, *INCOSE International Symposium*, vol.6, no.1, pp.1158-1165, juillet 1996.

[17] Moccozet L., Tardy C., Opprecht W. & Michel L., “Gamification-based assessment of group work”. *Interactive Collaborative Learning 2013*, septembre 2013.

[18] Morissette J., “Manières de faire l’évaluation formative des apprentissages selon un groupe d’enseignantes du primaire : Une perspective interactionniste”, *Thèse de doctorat*, Université Laval, Québec, Canada, 2009.

[19] Samoladas I., Gousios G., Spinellis D. & Stamelos I., “The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation”, *International Federation for Information Processing (IFIP)*, vol.275, pp.237-248, novembre 2008.

[20] Santos E., Campbell H., Patel D., Hindle A. & Amaral J., “Syntax and sensibility: Using language models to detect and correct syntax errors”, *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp.311-322, mars 2018.

[21] SC7, ISO/IEC FCD 25000, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE, ISO/IEC JTC1/SC7, mars 2014.

[22] Scallon G., *L’évaluation formative des apprentissages*, Presses Université Laval, Québec, Canada, 1988, 171p.

[23] Seaborn K. & Fels D., “Gamification in Theory and Action: A Survey”. *International Journal of Human-Computer Studies*, vol.74, pp.14-31, février 2015.

[24] Shahdatunnaim A., Noorminshah I. & Norasnita A., “Gamification in online collaborative learning for programming courses: A literature review”. *ARPN Journal of Engineering and Applied Sciences*, vol.10, no.23, pp.18087-18094, janvier 2015.

[25] Stevens S.S., “On the Theory of Scales of Measurement”, *SCIENCE*, vol.103, no.2684, pp.677-680, juin 1946.

[26] Swacha J. & Baszuro P., “Gamification-based e-learning Platform for Computer Programming Education”, *X World Conference on Computers in Education*, pp.122-130, juin 2013.

[27] Wajeeh D., “Evaluating Computer Programs: Tools and Assessment”, décembre 2006.

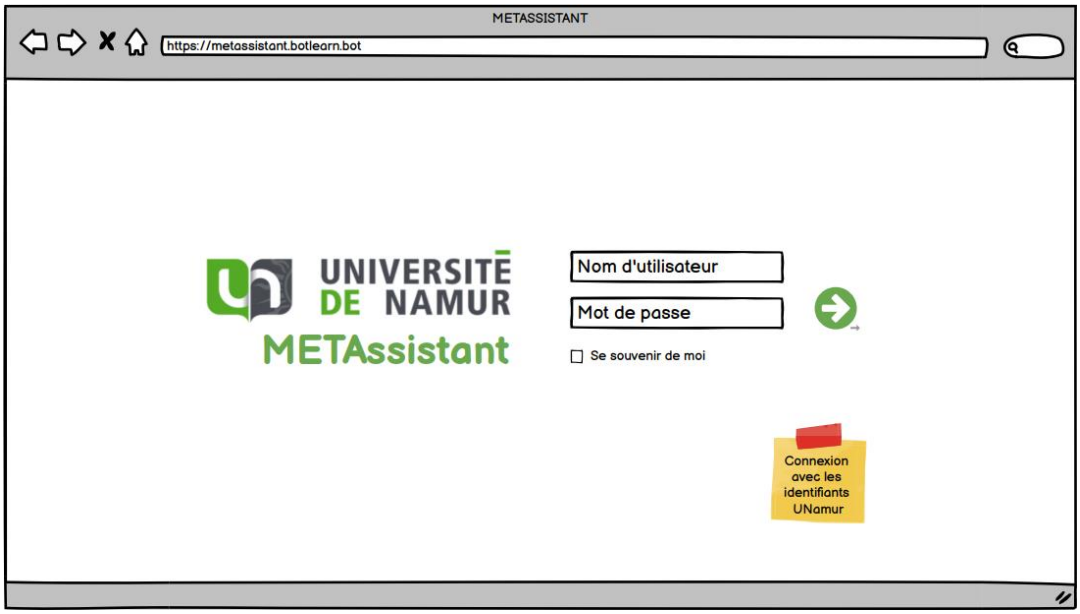
8 Annexes

Annexe 1 Maquettes du projet

Les maquettes ont été faites à l'aide de l'outil Balsamiq, plus précisément de sa version cloud.

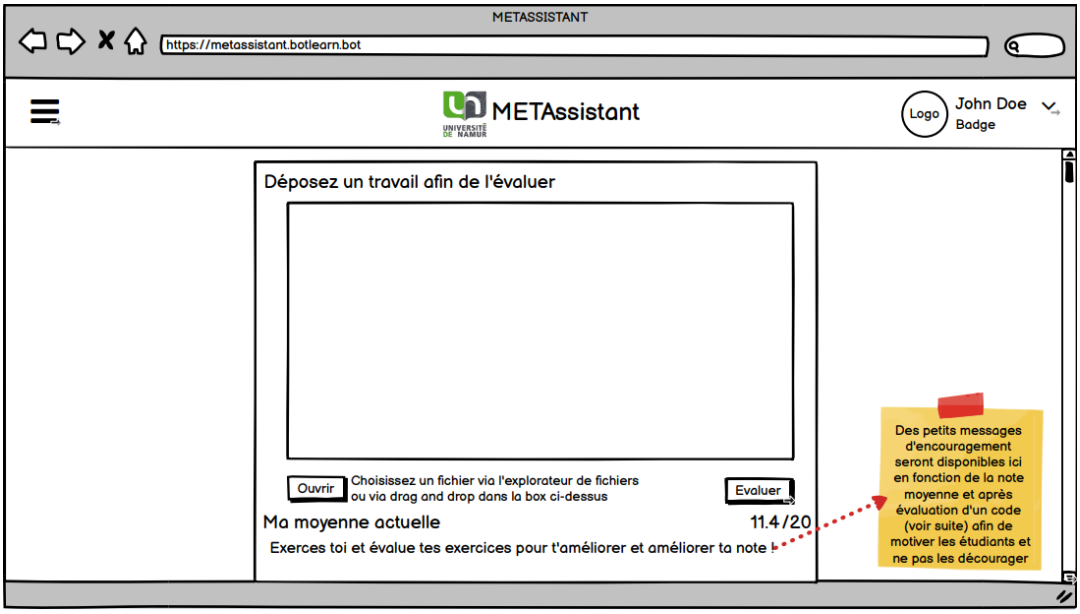
1.1 Connexion

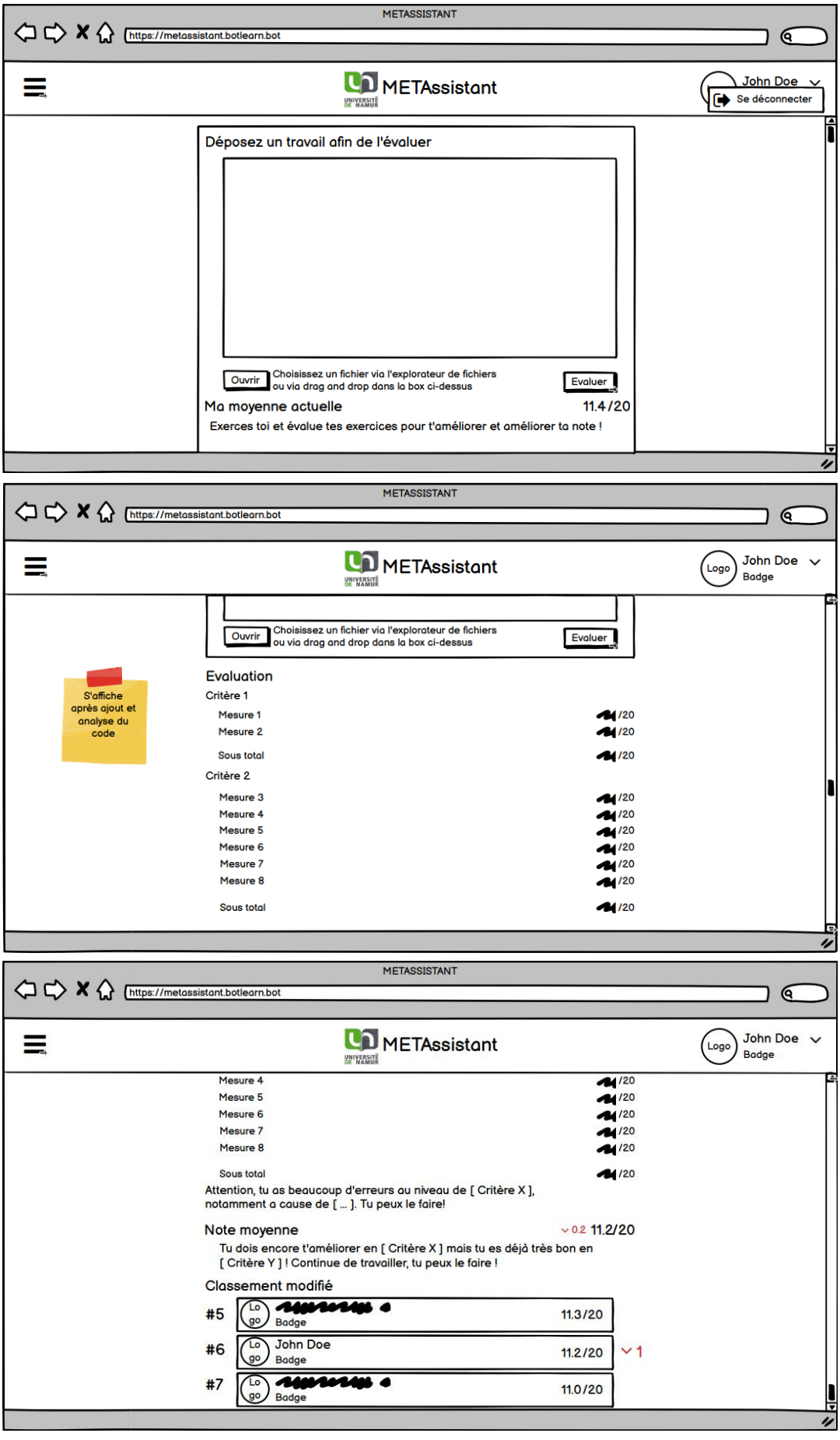
L'objectif est que la connexion se fasse avec les identifiants de l'UNamur. Les comptes étudiants et les comptes de professeurs auront des affichages différents. Les maquettes pour l'interface des étudiants sont décrites en 1.2, et celles pour les professeurs en 1.3.



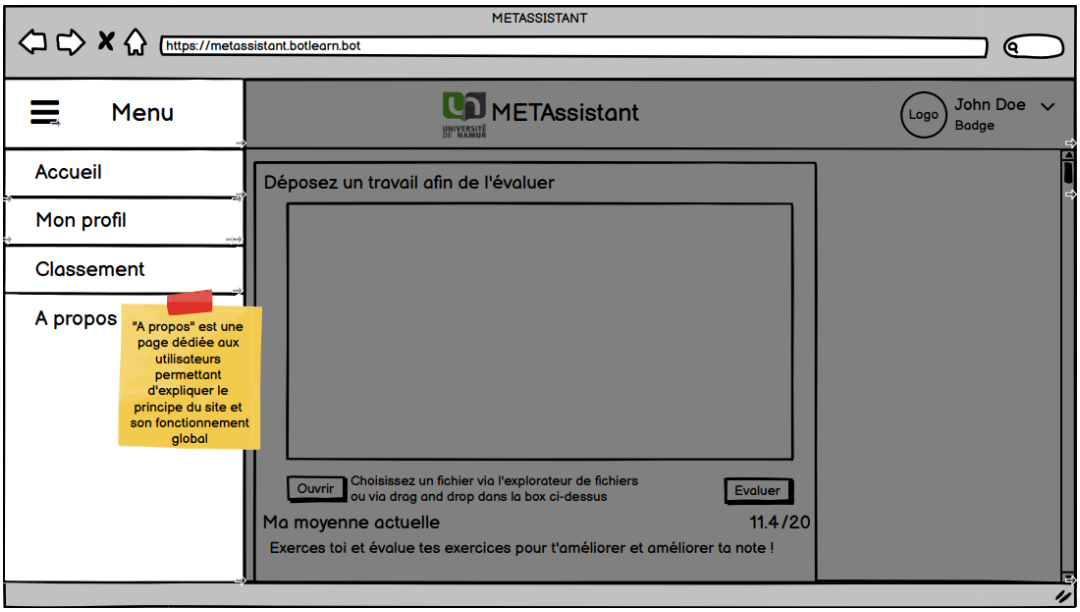
1.2 Interface utilisateur

1.2.1 Accueil

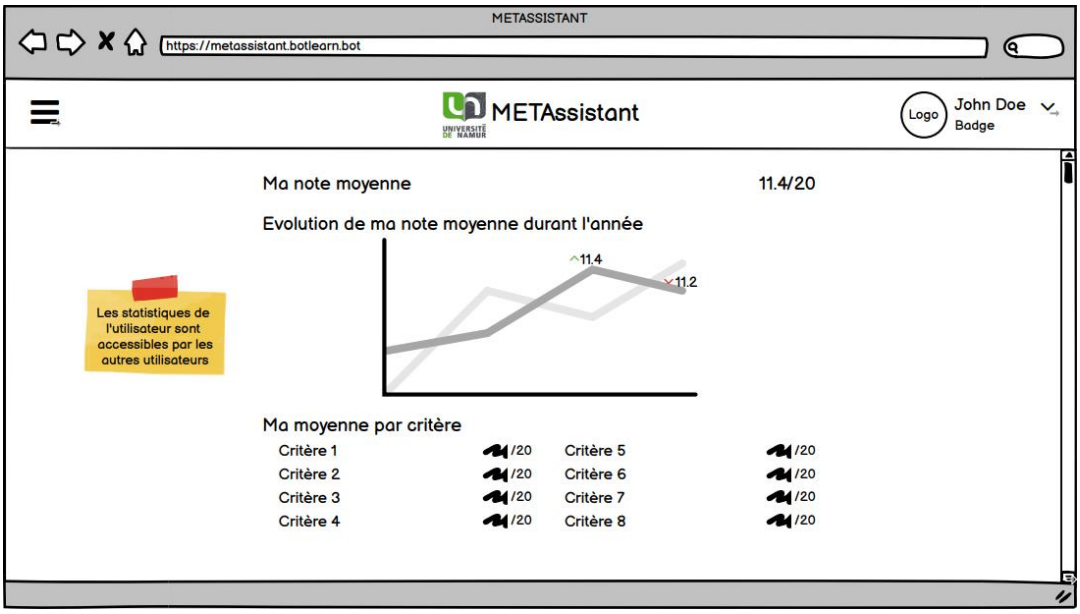


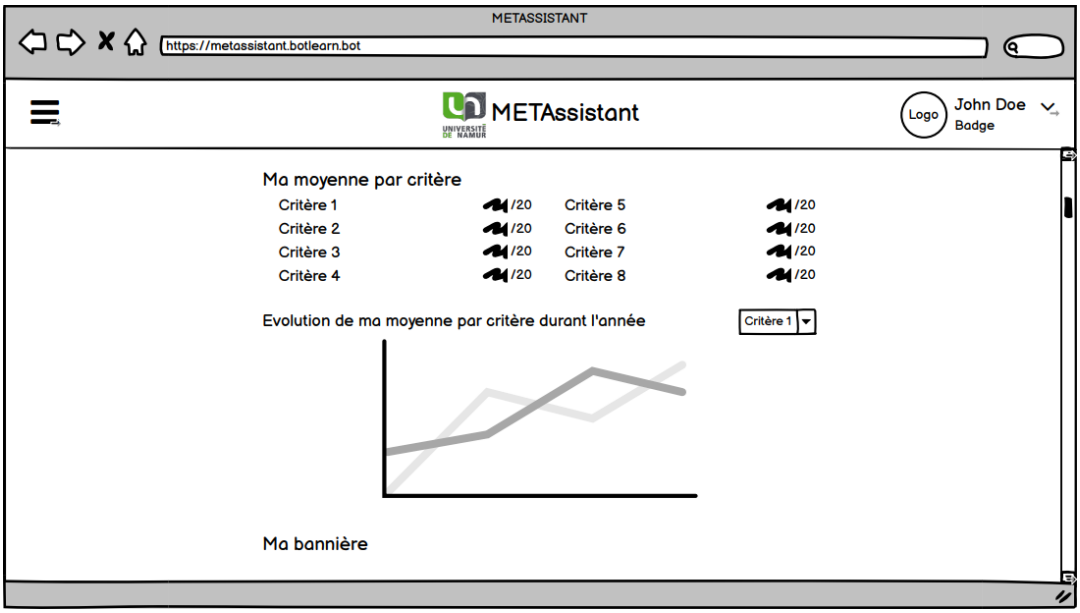

METASSISTANT
https://metassistant.botlearn.bot
METAssistant
John Doe
Se déconnecter
Déposez un travail afin de l'évaluer
Ouvrir
Choisissez un fichier via l'explorateur de fichiers
ou via drag and drop dans la box ci-dessus
Evaluer
Ma moyenne actuelle
11.4/20
Exerces toi et évalue tes exercices pour t'améliorer et améliorer ta note !
METASSISTANT
https://metassistant.botlearn.bot
METAssistant
Logo
John Doe
Badge
Ouvrir
Choisissez un fichier via l'explorateur de fichiers
ou via drag and drop dans la box ci-dessus
Evaluer
S'affiche après ajout et analyse du code
Evaluation
Critère 1
Mesure 1
Mesure 2
Sous total
Critère 2
Mesure 3
Mesure 4
Mesure 5
Mesure 6
Mesure 7
Mesure 8
Sous total
/20
METASSISTANT
https://metassistant.botlearn.bot
METAssistant
Logo
John Doe
Badge
Mesure 4
Mesure 5
Mesure 6
Mesure 7
Mesure 8
Sous total
/20
Attention, tu as beaucoup d'erreurs au niveau de [Critère X],
notamment a cause de [...]. Tu peux le faire!
Note moyenne
0.2
11.2/20
Tu dois encore t'améliorer en [Critère X] mais tu es déjà très bon en
[Critère Y] ! Continue de travailler, tu peux le faire !
Classement modifié
#5
Badge
11.3/20
#6
John Doe
Badge
11.2/20
1
#7
Badge
11.0/20

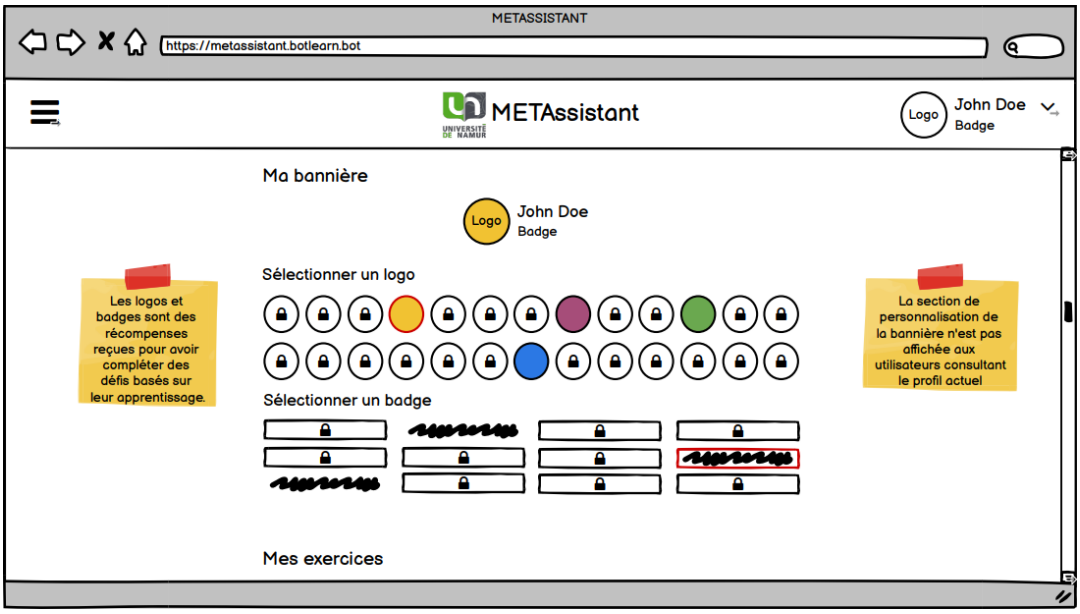
1.2.2 *Menu principal*

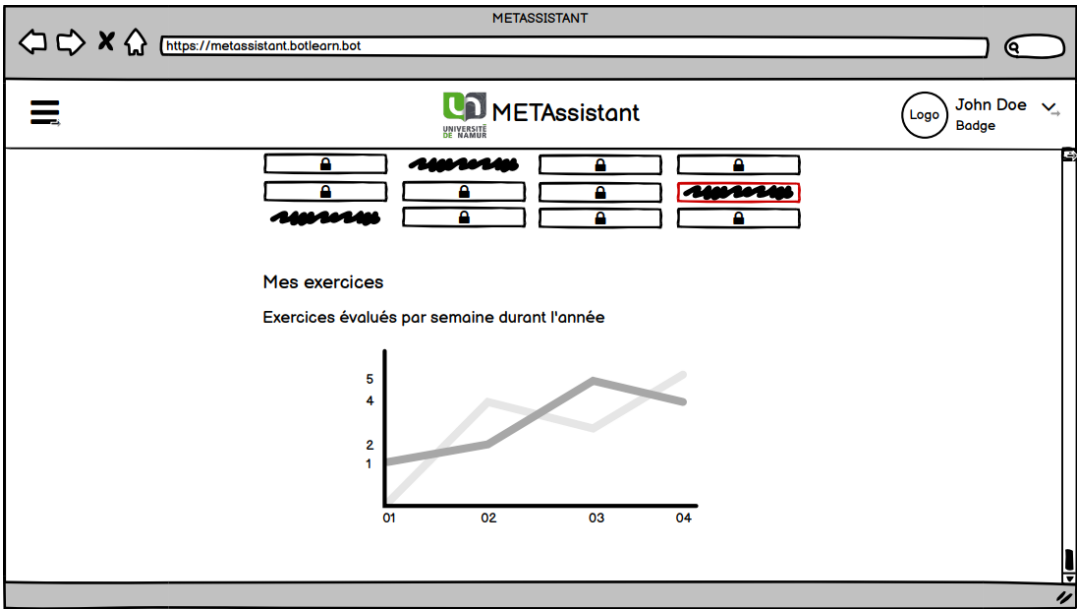


1.2.3 *Profil utilisateur*

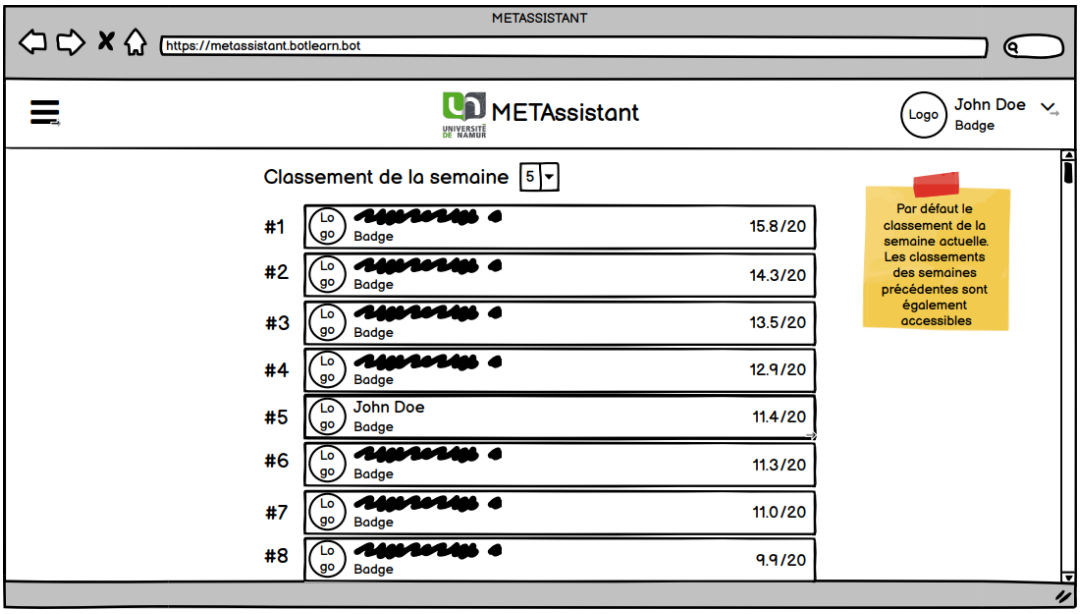



METASSISTANT
https://metassistant.botlearn.bot
METAssistant
Logo
John Doe
Badge
Ma moyenne par critère
Critère 1 /20
Critère 2 /20
Critère 3 /20
Critère 4 /20
Critère 5 /20
Critère 6 /20
Critère 7 /20
Critère 8 /20
Evolution de ma moyenne par critère durant l'année
Critère 1
Ma bannière


METASSISTANT
https://metassistant.botlearn.bot
METAssistant
Logo
John Doe
Badge
Ma bannière
Logo
John Doe
Badge
Les logos et badges sont des récompenses reçues pour avoir compléter des défis basés sur leur apprentissage.
Sélectionner un logo
La section de personnalisation de la bannière n'est pas affichée aux utilisateurs consultant le profil actuel
Sélectionner un badge
Mes exercices

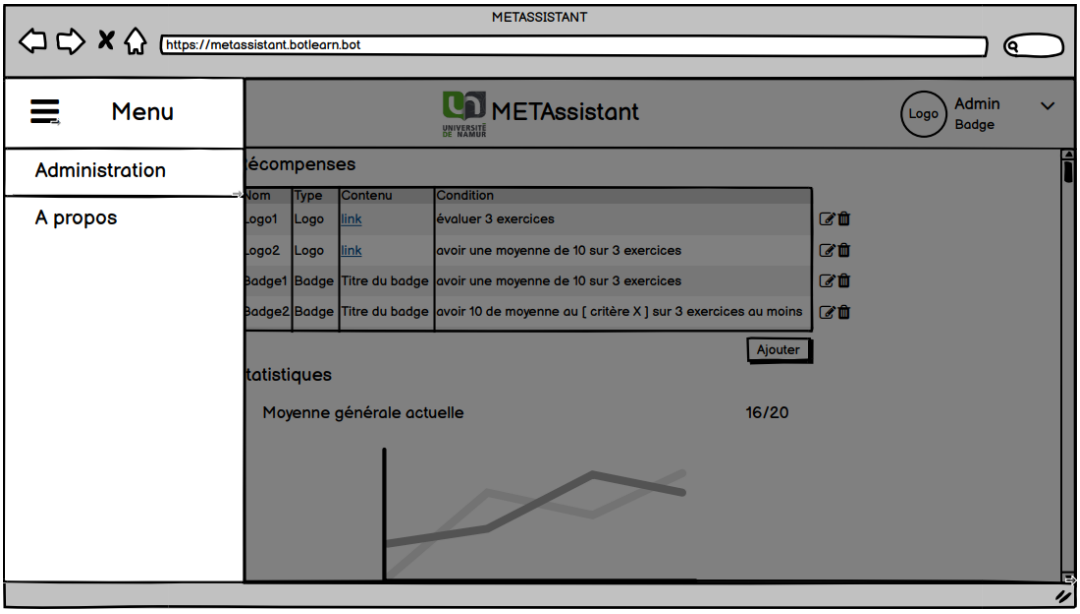

METASSISTANT
https://metassistant.botlearn.bot
METAssistant
Logo
John Doe
Badge
Mes exercices
Exercices évalués par semaine durant l'année
5
4
2
1
01
02
03
04

1.2.4 Classement



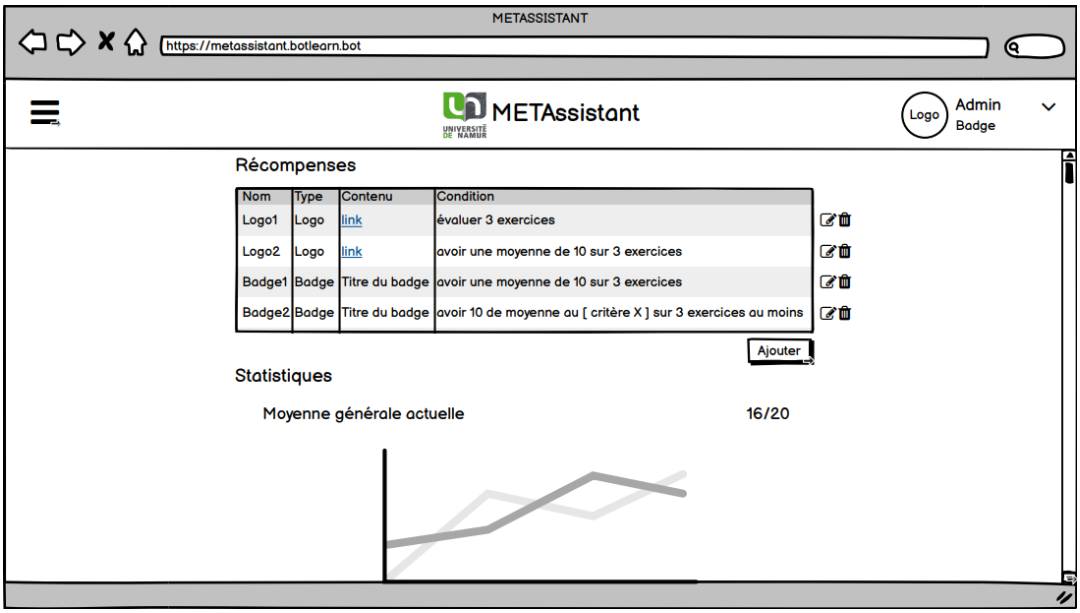
1.3 Interface administrateur

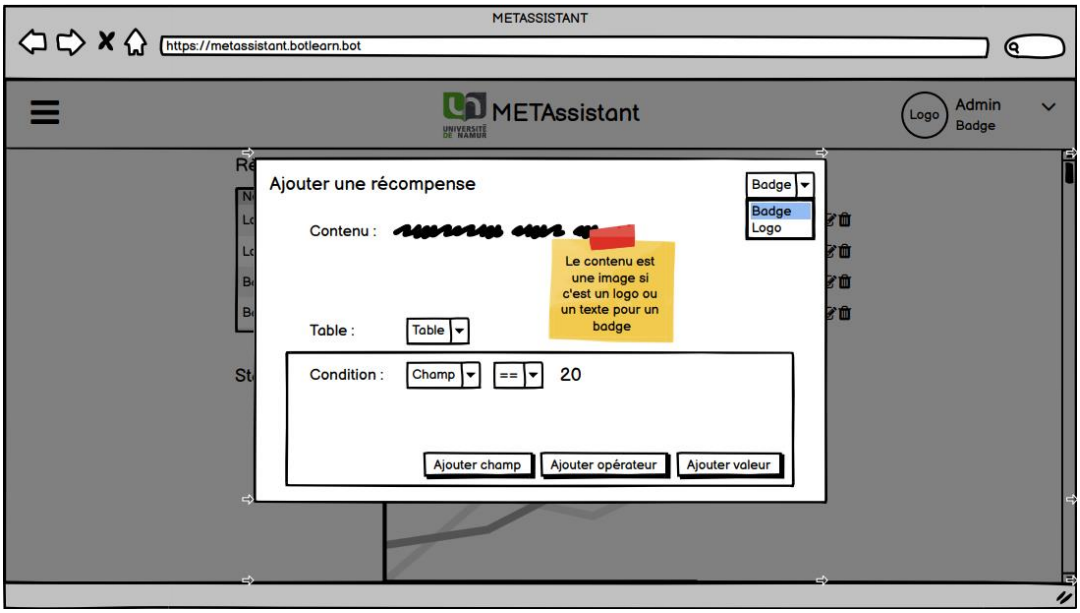
1.3.1 Menu principal



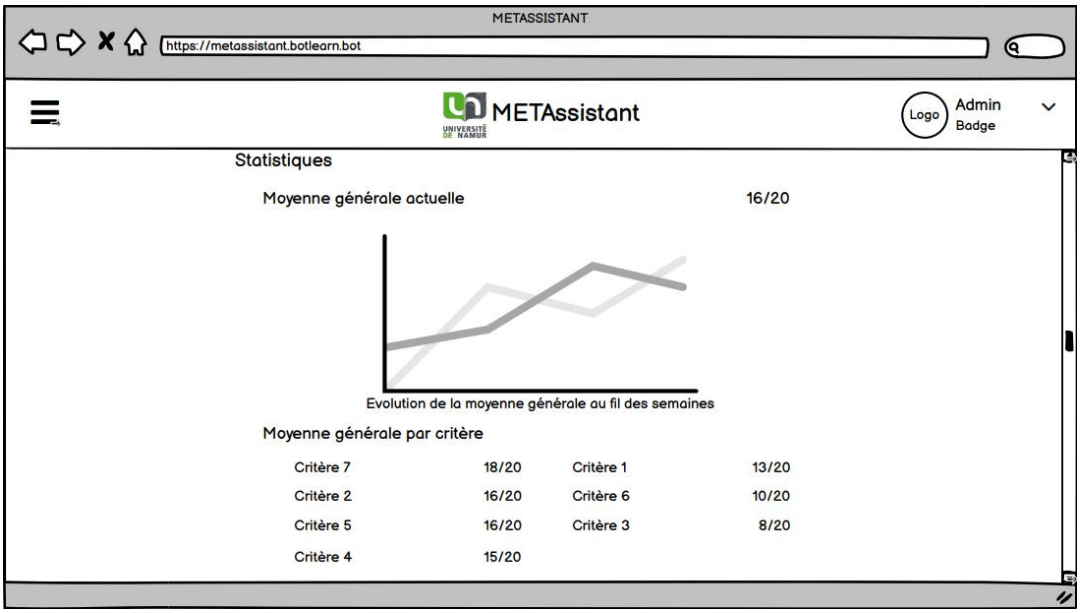
1.3.2 Page d'administration

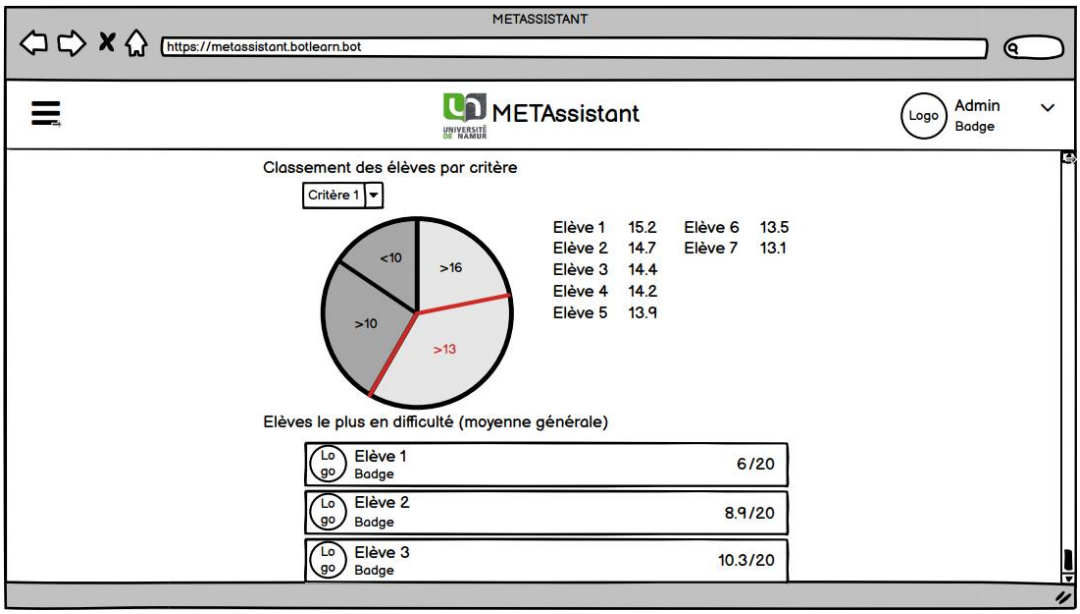
1.3.2.1 Gestion des récompenses





1.3.2.2 Statistiques générales


METASSISTANT
https://metassistant.botlearn.bot
METAssistant
Logo
Admin
Badge
Statistiques
Moyenne générale actuelle
16/20
Evolution de la moyenne générale au fil des semaines
Moyenne générale par critère
Critère 7 18/20
Critère 1 13/20
Critère 2 16/20
Critère 6 10/20
Critère 5 16/20
Critère 3 8/20
Critère 4 15/20


METASSISTANT
https://metassistant.botlearn.bot
METAssistant
Logo
Admin
Badge
Classement des élèves par critère
Critère 1
<10
>16
>10
>13
Elève 1 15.2
Elève 2 14.7
Elève 3 14.4
Elève 4 14.2
Elève 5 13.9
Elève 6 13.5
Elève 7 13.1
Elèves le plus en difficulté (moyenne générale)
Elève 1
Badge
6/20
Elève 2
Badge
8.9/20
Elève 3
Badge
10.3/20

Annexe 2 Description des tables de la base de données

2.1 Django_User

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_user	INTEGER	11	X	X	X	
login	VARCHAR	50	X	X		
password	VARCHAR	50	X			
mail	VARCHAR	100	X	X		
firstname	VARCHAR	50	X			
lastname	VARCHAR	50	X			

2.2 Student

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_user	INTEGER	11	X	X	X	
id_reward	INTEGER	4	X			Logo
id_reward_1	INTEGER	4	X			Badge

2.3 Reward

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_reward	INTEGER	11	X	X	X	

2.4 Logo

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_reward	INTEGER	11	X	X	X	
pic_url	VARCHAR	2000	X	X		

2.5 Badge

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_reward	INTEGER	11	X	X	X	
content	VARCHAR	100	X	X		

2.6 Ranking

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_ranking	INTEGER	11	X	X		
ranking_date	DATE	10	X			
average	DOUBLE	11	X			

number_ranking	INTEGER	11	X			
id_user	INTEGER	11	X	X		Student

2.7 Exercice

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_exercice	INTEGER	11	X	X	X	
content	VARCHAR	20 000	X			
note	DOUBLE	4	X			
post_date	DATE	10	X			
return_data	VARCHAR	8000	X			
id_user	INTEGER	11	X			Student

2.8 Measure

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_measure	INTEGER	11	X	X	X	
name	VARCHAR	200	X			
importance	INTEGER	2	X			
id_criterion	INTEGER	3	X			Criterion

2.9 Criteria

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_criterion	INTEGER	11	X	X	X	
name	VARCHAR	200	X			

2.10 Program_error

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_error	INTEGER	11	X	X	X	
error_code	VARCHAR	5	X	X		
error_statement	VARCHAR	200	X	X		
description	VARCHAR	300	X			
id_measure	INTEGER	4	X	X		Measure

2.11 Motivationnal field

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key

id_motivationnal_field	INTEGER	11	X	X	X	
sentence	VARCHAR	800	X	X		
type	VARCHAR	20	X			

2.12 Evaluation

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_measure	INTEGER	11	X	X	X	Measure
id_exercice	INTEGER	11	X	X	X	Exercice
note	DOUBLE	4	X			

2.13 Exercice_program_error

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_exercice	INTEGER	11	X	X	X	Exercice
id_error	INTEGER	11	X	X	X	Program_error
line	INTEGER	5	X			

2.14 Student_reward

Colonne	Type	Longueur	Obligatoire	Unique	Primary key	Foreign key
id_user	INTEGER	11	X	X	X	Student
id_reward	INTEGER	4	X	X	X	Reward
locked	BOOLEEN	1	X			

Annexe 3 Article soumis à EASEAI

L’article est accessible à partir de la page suivante.

A tool for evaluating computer programs from students

Quentin Vaneck
Fac. CS - NaDI/PReCISE
University of Namur
quentin.vaneck@gmail.com

Thomas Colart
Fac. CS - NaDI/PReCISE
University of Namur
colartthomas@outlook.com

Benoît Frénay
Fac. CS - NaDI/PReCISE
University of Namur
benoit.frenay@unamur.be

Benoît Vanderose
Fac. CS - NaDI/PReCISE
University of Namur
benoit.vanderose@unamur.be

ABSTRACT

Computer science studies are more and more popular, and teachers must face and adapt to the increasing number of students. Whereas small groups allowed more interactions between teachers and students, the resulting overcrowding takes away closeness and forces teachers to spend less time with each student. Therefore, the student can quickly feel submerged and helpless against the difficulty of the course. This paper proposes a solution that aims to reduce drop-out in programming courses. It offers an accurate feedback on the quality of students' Python code to deepen their understanding, together with a playful interface to boost their interest in programming. This solution developed under the name of "METAssistant" has two objectives. It allows students to use it to evaluate their programs and to get an accurate feedback, and teachers to have an overview of the understanding of the matter by their students.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Student assessment**; **Computing literacy**; • **Human-centered computing** → *HCI theory, concepts and models.*

KEYWORDS

Automated grading, Self assessment, Software quality, Static analysis, Gamification

ACM Reference Format:
Quentin Vaneck, Thomas Colart, Benoît Frénay, and Benoît Vanderose. 2021. A tool for evaluating computer programs from students. In *Proceedings of The 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Computer programs quality assessment is crucial in the context of automated grading of assignments and students' self-assessment. Several research works propose methods to assess a program based on predefined tests, but some also suggest to do so through quality measurement. Developed as part of a master thesis at the University of Namur, this paper presents a platform that relies on quality assessment methods from the ISO/IEC 25000 standard, and introduces complementary gamification principles for educational purposes. The paper addresses the following research questions:

- RQ1. How can we use static code analysis tools to automate program grading and feedback, even when the code may be non-executable and may not respect Python syntax?
- RQ2. How can we prompt students to improve their programming skills through continuous training?

ESEC/FSE 2021, 23 - 27 August, 2021, Athens, Greece

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Pros and cons for test-based evaluation

Pros	Cons
It can check the dynamic aspect of a program, i.e. the behaviour and correct functioning.	It does not consider as many elements as a teacher would such as good practices (conventions, readability etc.).
It always provide a relevant feedback within the scope of the exercise.	Tests must be planned in advance for each exercise.

2 RELATED WORKS

Several works in the literature are related to test-based and quality-oriented program evaluation methods and gamification.

2.1 Various evaluation methods

There exist several ways to assess code quality to provide feedback to its author. The following describes the main categories of methods based on predefined tests and quality criteria.

2.1.1 Test-based evaluation. Many automated self-assessment or grading tools are currently based on the use of test cases. INGInious is such a student code evaluation platform created at UCLouvain [1]. The goal of INGInious is to provide feedback on a code submitted by a student in the context of a programming exercise. These exercises and tests are predefined in the platform, so the student must (and can only) submit a code related to a specific exercise. Then, the platform will use unit tests to test the student's code and provide a feedback. The code provided by the student is executed with input values determined by INGInious and the output values are checked. The provided feedback depends on the context, it will not be the same for a recurring error in a particular exercise, or for an error that can occur in all types of exercises. Where a compiler may not be able to differentiate between these different cases, INGInious uses a static check to recognize the most frequent errors and return the information to the user. Table 1 summarises some pros and cons of such test-based evaluation platforms.

Table 2: Pros and cons for quality-oriented evaluation

Pros	Cons
Automated process, no need to explicitly predefine tests for each exercise. Considers many different criteria for a more precise evaluation. Based on a reputed standard made by experts.	Defining all measures can be a long and tedious task.

2.1.2 Quality-oriented evaluation. Quality-oriented evaluation consists in using software measurement to assess a set of quality criteria. The relevance of these quality criteria varies depending on the context and the purpose of the code being evaluated. In order to provide consistency regarding these criteria, international standards have been defined. ISO/IEC 25000 is such a standard and aims to provide a framework for the evaluation of software product quality. It defines every step of the process and gives examples to get started [2]. A measurement information model [3] clarifies how to implement a measurement plan.

Some research works use the above standards to evaluate source code quality. They consider quality criteria and a method of defining measurements. For example, SQO-OSS offers a metric-based program evaluation model with minimal human intervention, specifically for open source software [4]. The evaluation process is divided into two phases: the definition of the evaluation model and the definition of the aggregation method. In the first phase, the evaluation model is defined on the basis of quality criteria and associated metrics. In the second phase, the aggregation method represents the collection of data and the aggregation of measurement results in order to determine the quality of a code. Table 2 summarises some pros and cons of quality-oriented evaluation platforms.

2.2 Gamification

Gamification applies game and entertainment mechanics to make a task more attractive [5]. Making a situation more playful for a user has several objectives. Users might find themselves more engaged in a task if they have a goal, an objective to achieve, etc. To increase user engagement, one may define rewards for achieved objectives. This motivates the user to give his best for the task. The social aspect is also important. Contacts with other users allow cooperation and mutual support, which gives everyone the opportunity to get involved and sometimes find innovative solutions. The competitive aspect, with a ranking, the reward completion… also has beneficial effects as it can motivate a user to progress to be the best [6].

Codewars is such a platform. It allows its users to learn and deepen their programming skills through gamification. It proposes a large set of exercises and a customizable user profile with stats and rewards, as well as a progression bar and a global ranking [1].

[1] Codewars platform (2021) https://www.codewars.com/

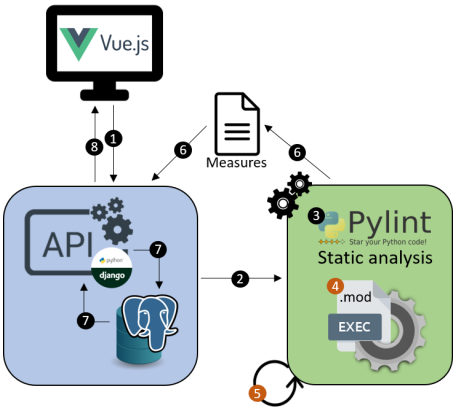


Figure 1: System architecture and program evaluation path.

3 PROPOSED APPROACH

The solution proposed here opts for a quality-oriented (rather than test-based) source code assessment, as the evaluation and the feedback can be very precise, with gamification aspects.

3.1 Architecture

Figure 1 represents a basic view of the platform's architecture. The architecture can be divided in three blocks: the user interface, the API and the assessment algorithm. The user interface (computer screen symbol) is written in Vue.js and integrates aspects of gamification as described in Section 3.3. The assessment algorithm (green block) is explained in more details in Section 3.2. Then, the API and data access (blue block) allows a user to retrieve and update data from the PostgreSQL database, assess their program and get a feedback. The API is implemented with the Django framework and follows REST conventions. The assessment process shown in Figure 1 through action numbering is described in Section 3.1.1.

3.1.1 Execution path for quality assessment. Figure 1 also shows an execution path, with steps numbered from 1 to 8, from the moment code is submitted up to feedback retrieval. Notice that orange numbered elements define a conditional execution only used when an error occurs in the assessment process.

(1) The student's code to assess is sent to the API
(2) The program is submitted to the assessment algorithm
(3) A static code analysis, configured according to the program, is processed thanks to the Pylint Python library in order to retrieve error codes that are relevant for the assessment of the user's program.
(4) If a fatal error occurs at step 3 (i.e., the code analysis tool encountered a problem while processing), the error line number is sent to the line correction algorithm (LCA) described at section 3.2.3.
(5) If the error has been successfully corrected by the LCA, the process restarts at step 3.
(6) When Pylint succeeds, error codes are extracted to evaluate predefined measures and the result is sent back to the API.
(7) Data are stored in the database.
(8) The feedback is returned to be displayed to the user.

3.2 Quality assessment

This section explains the tools and the algorithms used.

3.2.1 Building and evaluating measures. The quality evaluation process used here follows the measurement information model explained in Section 2.1.2 [3]. Therefore, the main focus is to define relevant measures to evaluate each criteria from the ISO/IEC 25000 standard. At first, base measures are collected from source code attributes. An example of a base measure is the number of lines of code or the number of errors from the static analyser. Base measures are raw numbers gathered from the code but do not convey much quality information by themselves [7]. Therefore, with a predefined function, base measures are transformed into more relevant measures. For example, it can be the number of errors per line. This derived measure has to be associated to the corresponding criterion. The number of measurements increases the accuracy of the evaluation, as each measurement corresponds to a specific aspect of the source code. The value obtained after the measurement is then interpreted using indicators to assign a note for the concerned measure.

3.2.2 Pylint. Pylint is a static code analysis tool for Python that checks five types of errors: "convention" for programming standard violations, "warning" for Python-specific issues, "error" for elements that are likely bugs, "refactor" for bad code smells and "fatal" for errors preventing execution. This tool is highly configurable and specific errors can be disabled, if needed. There also exists a library to invoke Pylint programmatically [2]. This is a great tool to check good practices in code and to efficiently assess syntactic measures notably in functionality, maintainability and security criteria.

3.2.3 Line correction algorithm (LCA). Pylint being a classical static code analyser, it lacks flexibility and can easily get stopped in its execution. Let us consider a source code where the Pylint process results in a fatal error at a given line. The goal of the LCA is to use Python's syntax and pattern matching to understand the user's intent and to try to refactor the error line to match it.

To do this, the first step is to redefine Python's syntax with (1) all the symbols and reserved keywords and (2) regular expressions to match elements of Python's syntax (e.g., expressions, instructions, definitions, etc.). This will be useful to compare and match elements from the user's error line. To do so, it is necessary to build the regular expression of the error line by considering the previously defined syntax. The algorithm looks for the matching symbol or regular expression for each element of the error line, using regular expression matching as well as the Damerau-Levenshtein distance explained in Section 3.2.4 to make sure to find the right symbol. This distance offers a margin of error in particular to prevent typos.

When the error line's regular expression is built, it tries to match a regular expression from Python syntax considering each element of the error line's regular expression. Then, a sorted list of the best matches is returned and is used to refactor the error line. Refactoring is done by rearranging elements in the good order considering the matched patterns and by correcting erroneous elements.

3.2.4 Damerau-Levenshtein (D-L) distance. The Damerau-Levenshtein distance calculates the edit distance between two strings. It is an extension of the Levenshtein distance and is useful especially for typos detection. As well as counting the minimum number of operations needed to transform a string into another, an operation being an insertion, deletion or substitution, it also considers the transposition of characters (i.e., a swap between two letters) [9]. As it does not consider the length of both the compared words, a normalized version of D-L allows one to compute the ratio of the distance to the length of the longer string. The result belongs in $[0, 1]$, where 0 means that no differences were found and 1 that the two words are opposite, with no similarities [3].

Table 3: Results from the comparison of two values using different algorithms

Algorithm	Value 1	Value 2	Result
Levenshtein	isinstance	isnistance	2
	from	form	2
Damerau-Levenshtein	isinstance	isnistance	1
	from	form	1
Normalized D-L	isinstance	isnistance	0.1
	from	form	0.25

Table 3 shows results from those algorithms on two samples where a transposition occurred. The Levenshtein distance counts two operations between the compared codes (deletion + insertion), whereas the D-L distance counts the transposition as one. The D-L distance is more relevant here. Additionally, the normalized D-L distance considers the length and gives an easier-to understand indication on the (rate of) similarity of the compared words.

The solution proposed in this paper uses a Python library [4] to calculate the normalized D-L edit distance to compare the elements of a given error line to Python's symbols and keywords. Based on the result, the checked word can be considered as relevant or not.

3.3 Gamification

The proposed system offers several gamification aspects. Firstly, the platform allows users to see different statistics. On their profile, students can see their average score and how it has evolved over time. It is also possible to look at the different averages for each criterion that make up an exercise. All these elements allow the student to see where he stands in the learning process, to see where he is lacking, where he has improved… and therefore to satisfy a possible need for recognition as other students can also see those statistics. Secondly, it introduces competitiveness through a league table between students. This ranking shows the students with the best average in descending order. It allows students to situate themselves in relation to others and therefore motivates them to practice more exercises in order to climb the ranking. The last aspect of gamification is the notion of rewards. Depending on his progress, the number of completed exercises, his average, etc., the student will be given certain rewards such as badges or logos to customize his profile, which will be visible to other students in the

[2] Pylint Python library (2021) https://pypi.org/project/pylint/

[3] Geoffrey Fairchild (2021) *pyxDamerauLevenshtein* https://github.com/gfairchild/pyxDamerauLevenshtein

[4] pyxDamerauLevenshtein Python library (2021) https://pypi.org/project/pyxDamerauLevenshtein

```
1  import os
2  import sys
3  import random as random
4
5  df write_random_pow(number):
6      r = random.randint(0, 10)
7      random_pow = pow(number, r)
8
9      "Random pow: " = s
10
11     with os.open('random_pow.txt', None, 'w') as file:
12         file.write(s + random_pow)
13
14     return random_pow
```

Figure 2: Example of student code submitted for evaluation.

Table 4: Evaluation details for code at Figure 2.

Criteria	Measure	Note (/20)
Functionality	syntax accuracy	20
	fatal errors	16.6
Maintainability	variable relevance	16.9
	import conventions	10
Total score:		15.87

league table or the profile page. Through this system, a certain need for completion is introduced as the student will want to collect all possible awards in the platform. This aspect is reinforced through the use of loading bars that allow students to situate themselves.

4 ILLUSTRATION

Figure 2 presents a sample code given to the system for evaluation. Errors have been deliberately inserted for the example. At first, running Pylint will result in an error. Actually, errors at line 5 and 9 are syntactic errors that any static analyser will not understand, those are the so called fatal errors. Each will be sent to the inline correction algorithm that will suggest a correction considering the detected intent behind the line. At line 5, it will detect the intent to define a function, and will replace the typo "df" by "def". At line 9, it will detect the intention to make an assignation and will reorder the line with the variable at the beginning and his value at the end.

Those two lines being adjusted, Pylint can now start analysing the content and return error codes related to the given program. Errors at line 2 and 3 for unused import and same alias name, or at line 6 and 9 for invalid variable names, as well as the two fatal error encountered, are being considered for the evaluation.

Table 4 summarises the evaluation details for each measure. Notice that, on a program, only the concerned measures are considered for the final score, which is the average of the above measures. In that way, even if there are errors at some points, considering multiple aspects of programming is less punitive and rewards the student for his efforts. It is important for beginners to have the details for each measure so they know what they did wrong and what was nice, it is more motivating for them. To be complete in the feedback, the specific errors detected are displayed to the student so that he knows exactly why he has the given score.

It is also interesting to note that there is an error at line 12, trying to concatenate a string with an integer, that will not be detected because of the use of a static code analyser as it is a semantic error.

5 CONCLUSION AND FUTURE WORKS

In this paper, we first described the main methods to assess the code of a student. Then we explained our solution "METAssistant" and applied it on a small sample to illustrate. The goal is to provide the most relevant feedback possible to help students deepen them knowledge. This feedback considers different measures, related to quality criteria provided by the ISO/IEC 25000 standard, defined to assess the most accurately possible the students' programs, taking into account every aspects of programming. The solution also provides comprehensive feedback to the student and aims to boost interest and engagement of its users through a gamified platform. However, the current system is not optimal. It offers a basis to assess a student's code, but it can be improved at different levels.

Firstly, the defined measures have to be refined and completed. The system currently implements a set of measures mainly for functionality, maintainability and security criteria, for the most thanks to Pylint. But it lacks of semantic analysis and many other measures. Those have to be enhanced and completed by measures considering the ISO/IEC 25000 standard and potentially using other tools to deepen the analysis.

Secondly, the gamification aspect can be pushed further. Other gamification features can help to increase the student's implication and learning capability.

Finally, the proposed LCA for fatal errors can be assimilated to an interpreter with the matching of a created regular expression from the error line and the syntax regular expression. Even if the matching is improved by some tools like the Damerau-Levenshtein distance and the matching algorithm, it is still lacking precision to find the good error line number or to correct multi-line errors. Only a limited set of errors can be currently handled. A solution could be to use a "long short-term memory" (LSTM) architecture of "recurrent neuronal network" (RNN). Some research has already been made on this topic for the Java language [10].

REFERENCES

[1] Anthony Gego, Benjamin Frantzen, Guillaume Derval, Peter Van Roy, Pierre Reinbold (2020). *Automatic grading of programming exercises in a MOOC using the INGInious platform.* Catholic University of Louvain, Belgium.

[2] ISO/IEC 25000 series of standards (2014). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE.* https://www.iso.org/fr/standard/64764.html

[3] Alain Abran, Rafa Al Qutaish, Jean-Marc Desharnais, Naji Habra (2005). *An Information Model for Software Quality Measurement with ISO Standards*

[4] Ioannis Samoladas, Ioannis Stamelos, Diomidis Spinellis, Georgios Gousios (2008). *The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation.*

[5] Deborah Fels, Katie Seaborn (2015). *Gamification in theory and action: A survey.*

[6] Dan Dixon, Lennart Nacke, Rilla Khaled, Sebastien Deterding (2014). *Du game design au gamefulness : définir la gamification.*

[7] Norman Fenton, James Bieman (2014). *Software metrics: a rigorous and practical approach.* CRC press.

[8] S.S. Stevens (1946). *On the Theory of Scales of Measurement.* SCIENCE, vol. 103, no. 2684, pp. 677-680.

[9] Viny Christanti M., Rudy, Dali S. Naga (2018). *Fast and Accurate Spelling Correction Using Trie and Damerau-levenshtein Distance Bigram.* TELKOMNIKA, vol.16, no.2, pp. 827-833.

[10] Abram Hindle, Dhvani Patel, Eddie A. Santos, José N. Amaral, Joshua C. Campbell (2018). *Syntax and sensibility: Using language models to detect and correct syntax errors.*